PCT WORLD INTELLECTUAL PROPERTY ORGANIZATION
International Bureau

INTERNATIONAL APPLICATION PUBLISHED UNDER THE PATENT COOPERATION TREATY (PCT)

(51) International Patent Classification 6 :
G06F 15/00, H04M 1/64

A1

(11) International Publication Number: **WO 96/13007**

(43) International Publication Date: 2 May 1996 (02.05.96)

(21) International Application Number: PCT/US95/12630

(22) International Filing Date: 23 October 1995 (23.10.95)

(30) Priority Data:
08/327,704 24 October 1994 (24.10.94) US

(71)(72) Applicant and Inventor: ROSSIDES, Michael, T. [US/US]; 3666 Upton Street, N.W., Washington, DC 20008 (US).

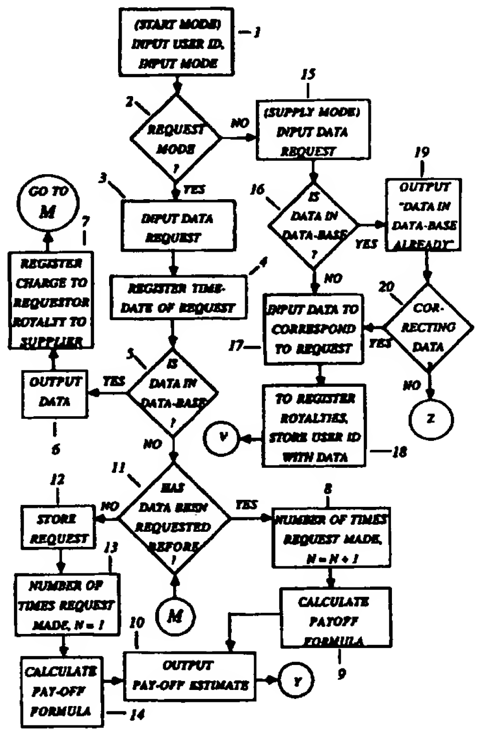
(81) Designated States: AM, AT, AU, BB, BG, BR, BY, CA, CH, CN, CZ, DE, DK, EE, ES, FI, GB, GE, HU, JP, KE, KG, KP, KR, KZ, LK, LR, LT, LU, LV, MD, MG, MN, MW, MX, NO, NZ, PL, PT, RO, RU, SD, SE, SI, SK, TJ, TT, UA, UZ, VN, European patent (AT, BE, CH, DE, DK, ES, FR, GB, GR, IE, IT, LU, MC, NL, PT, SE), OAPI patent (BF, BJ, CF, CG, CI, CM, GA, GN, ML, MR, NE, SN, TD, TG), ARIPO patent (KE, LS, MW, SD, SZ, UG).

Published
With international search report.

(54) Title: A DATA COLLECTION AND RETRIEVAL SYSTEM FOR REGISTERING CHARGES AND ROYALTIES TO USERS

(57) Abstract

A self-organizing data-base that charges (7) users who find data in it and pays (7) users who supply the data found. Has a built in feedback mechanism, called the Pay-off Meter (11, 14), that tells users what data needs supplying based on the number of requests (13) for that data over time. The Pay-off Meter outputs a projected Pay-off (10) for supplying the data.


(START MODE) INPUT USER ID, INPUT MODE
1
2
REQUEST MODE ?
NO
YES
15
(SUPPLY MODE) INPUT DATA REQUEST
19
OUTPUT "DATA IN DATA-BASE ALREADY"
GO TO M
3
7
INPUT DATA REQUEST
16
IS DATA IN DATA-BASE ?
YES
NO
REGISTER CHARGE TO REQUESTOR ROYALTY TO SUPPLIER
4
REGISTER TIME-DATE OF REQUEST
20
COR-RECTING DATA
YES
NO
INPUT DATA TO CORRESPOND TO REQUEST
17
5
IS DATA IN DATA-BASE ?
YES
NO
OUTPUT DATA
6
TO REGISTER ROYALTIES, STORE USER ID WITH DATA
18
V
Z
11
HAS DATA BEEN REQUESTED BEFORE ?
12
NO
YES
8
STORE REQUEST
NUMBER OF TIMES REQUEST MADE, N = N + 1
13
NUMBER OF TIMES REQUEST MADE, N = 1
M
CALCULATE PAYOFF FORMULA
9
10
OUTPUT PAY-OFF ESTIMATE
Y
CALCULATE PAY-OFF FORMULA
14

FOR THE PURPOSES OF INFORMATION ONLY

Codes used to identify States party to the PCT on the front pages of pamphlets publishing international applications under the PCT.

AT	Austria	GB	United Kingdom	MR	Mauritania
AU	Australia	GE	Georgia	MW	Malawi
BB	Barbados	GN	Guinea	NE	Niger
BE	Belgium	GR	Greece	NL	Netherlands
BF	Burkina Faso	HU	Hungary	NO	Norway
BG	Bulgaria	IE	Ireland	NZ	New Zealand
BJ	Benin	IT	Italy	PL	Poland
BR	Brazil	JP	Japan	PT	Portugal
BY	Belarus	KE	Kenya	RO	Romania
CA	Canada	KG	Kyrgystan	RU	Russian Federation
CF	Central African Republic	KP	Democratic People's Republic of Korea	SD	Sudan
CG	Congo			SE	Sweden
CH	Switzerland	KR	Republic of Korea	SI	Slovenia
CI	Côte d'Ivoire	KZ	Kazakhstan	SK	Slovakia
CM	Cameroon	LI	Liechtenstein	SN	Senegal
CN	China	LK	Sri Lanka	TD	Chad
CS	Czechoslovakia	LU	Luxembourg	TG	Togo
CZ	Czech Republic	LV	Latvia	TJ	Tajikistan
DE	Germany	MC	Monaco	TT	Trinidad and Tobago
DK	Denmark	MD	Republic of Moldova	UA	Ukraine
ES	Spain	MG	Madagascar	US	United States of America
FI	Finland	ML	Mali	UZ	Uzbekistan
FR	France	MN	Mongolia	VN	Viet Nam
GA	Gabon				

A Data Collection and Retrieval System For Registering Charges and Royalties to Users

Field of the Invention

The invention relates to a system for collecting and retrieving data and charging users for finding the data and paying users for supplying the data.

Background--The Prior Art

The parent disclosed a new type of data-base system. The novelty of the parent system centers around a loop of sorts in which the system registers information about the demand for given data. From this demand information, the system calculates a Pay-off Estimate that tells users, especially the Requestors of the data, the projected amount of income they might receive for entering the data into the system. Then once the data is entered, users who request it are charged and royalties are paid to the supplier.

The goal of the parent was to describe this loop. The loop is a quite basic and can be built upon. This continuation then adds new matter to the parent in three areas. First, it describes a new feature for displaying the pay-off estimates of certain kinds of data. Second, it describes the collection of more information about the demand for a piece of data, particularly price information. Third, it describes a form of the invention not envisioned in the parent, in which the system does not actually find and output data (answers) but finds and outputs pointers or reference information telling where the data can be found.

The specification of the PCT version of the parent has been copied into this continuation. Thus much of the text is basically identical with the parent.

Summary

The SODB is a data-base system that charges users who find data in it and pays users who supply the data found. It can be a single machine or a network. The key to the SODB is a built in feedback mechanism, called the Pay-off Meter, that tells users what data needs supplying based on the number of requests for that data. When a person requests a piece of data not in the SODB, the Pay-off Meter registers the request. Based on the rate of requests registered, this function estimates the royalties the data will generate once the data is supplied. The more requests in a period of time the greater the expected pay-off.

The key is that the expected pay-off is announced to each requestor of the data. The beauty is that each requestor may have to find the data anyway elsewhere. To collect the pay-off, a requestor has only to "call" the SODB back and input the answer. A sensitive and potent feedback loop is created insuring that the more a piece of data is requested, the more likely it will be supplied by a requestor or by someone a requestor tells of the pay-off. Moreover, this pay-off creates an incentive to correct or update faulty data.

Brief Description of the Drawings

Figure 1 shows a flow chart of a basic SODB.

Figures 2a shows the flow chart of the Request Mode of a lowest price locator.

Figure 2b shows the flow chart of the Supply Mode of a lowest price locator.

Figures 3 shows a flow chart of the steps preceding the gathering of information on what users are willing to pay for given data.

Figure 3a shows a flow chart of steps for gathering of information on what users are willing to pay for given data.

Figure 3b shows another set of steps for gathering of information on what users are willing to pay for given data.

Figure 3c shows another set of steps for gathering of information on what users are willing to pay for given data.

Figure 3d shows another set of steps for gathering of information on what users are willing to pay for given data.

Description

The SODB is a data-base system that charges users for the data they receive and pays royalties to users who input that data. What differentiates the SODB from other data-bases is a function that tells users who request data what the estimated royalty value is for supplying the data. The function keeps track of the rate of requests for the data and from this rate projects a demand rate. The estimated demand multiplied by the royalty rate yields a projected royalty stream. If a person requests a piece of data that is not in the SODB, the SODB outputs the projected value of inputting the data. (If the person finds the data answer is in the SODB, the SODB still can output the projected royalties for improving, correcting or updating the data.) Then, if necessary, the SODB tells users how to input the data. In sum, the SODB is a powerful feedback system that tells users what data needs supplying, tells them the financial incentive to supply it, tells them how to supply it and then pays them for supplying it.

Some Definitions and Comments

Start Mode: The procedure the SODB executes to allow users to access the SODB and choose between the Request and Supply Modes. Start Mode is not strictly necessary as long as its "logging on" functions are part of the Request and Supply Modes.

Request Mode: The procedure the SODB executes to provide answers and/or Pay-off estimates to Requestors. In the Request Mode, a user inputs a question causing the SODB to search for the corresponding answer. If the answer is not found, a Pay-off Estimate is outputted. If the answer is found, the answer is outputted along with the Pay-off Estimate (see Pay-off Meter) and a Charge is registered to the user.

Supply Mode: The procedure the SODB executes to allow users to input answers, potential answers and raw data. User identification data is registered along with the inputted data so that the user can be credited with royalties each time the data is used to supply answers.

Requestor: User who accesses the SODB by Request Mode seeking an answer. The Requestor owes a Charge if the answer is found.

Supplier: User who enters the Supply Mode to input an answer or raw data. The Supplier gets paid a Royalty each time the answer or the raw data is used by th SODB as determined by the royalty rules of the SODB.

Check Mode: The procedure the SODB executes to allow users to check the Pay-off Estimate given data. In this mode, a user is neither a Requestor nor Supplier though the Check Mode could be accessed through either the Request or Supply Modes.

Charge: The amount owed by a Requestor who receives an answer from the SODB.

Royalty Rules: The rules, embodied in functions, that determine the amount due to a Supplier of an answer (or of the raw data that is necessary for an answer), each time that answer is outputted to a Requestor.

Payments Register: The function the SODB executes to register payments owed by Requestors and payments due Suppliers. When an answer is outputted, the Payments Register registers who is owed a royalty and who owes a charge for the data used. What the Payments Register registers depends on the royalty rules of the SODB.

Question: Specific data corresponding to other specific data called the answer. When entered into the SODB by a Requestor, causes the SODB to search for the corresponding answer.

Answer: Specific data corresponding to other specific data called the question. An answer may be static, for example, the chemical structure of gasoline does not change. It may be dynamic, for example, the price gasoline does change. And, it may be improvable as well, for example, the octane of gasoline may be more accurately given. An answer may be long or short. It may have one component or many. For example, the question, "What are the Chinese restaurants in Biloxi?" may yield one restaurant or many.

The Correspondence Between a Question and Its Answer: There can only be one answer for any question, though that answer may have many components. Of course, a question may have multiple, even infinite, answers. But, the SODB requires rules that limit the answers to one, in the following sense: the answer to a question must be a finite set of data that is outputted to the requestor and charged for. The answer is what the requestor pays for. (A big problem in defining "answer" is that no one has come up with a good definition yet.) In the SODB, users input the answers they deem best. And users police the accuracy of those answers. The SODB accepts untrue or approximate answers, for it cannot divine meanings and truth, but any answer is displaced by a better answer. A better answer is one that, by convention and by the rules of the SODB, satisfies a question better than the existing answer. A user may displace one answer with another. If there is a dispute between users as to which answer is better, a neutral third party, the **Data-Base Manager** can be alerted to settle the dispute.

Of course, with many types of questions, whether an answer is "better " is not clear by convention. There may be many, even an infinite number of equally good answers. So, depending on the type of question, the SODB rules must limit the possible answers. One rule, for example, may be that the first answer inputted is considered better than all equivalent answers. However, no set of rules can capture truth and therefore, the manager, has final authority to decide whether an answer is true or not and whether one answer is better than another. See also Quality Control Functions below.

Potential Answer: An answer that may become the best answer in the SODB to a question.

Raw Data: If it has the requisite functions, the SODB can process raw data to arrive at answers. A piece of raw data may itself be considered the answer to a question. For example, the question, "What is the closest McDonalds to 1234 Main Street?," might require the SODB to have map coordinates for 1234 Main Street. Therefore, the coordinates are raw data. And, the coordinates themselves are the answer to the question, "What are the coordinates of 1234 Main Street?." Any answer for one question may be raw data for answering other questions. (The distinction is largely artificial between data that is by itself an answer and "raw data" that is used in calculating an answer. The distinction is artificial because "raw data" usually answers at least one question by itself. We will keep this term for now but probably improve on it in a future application.)

Storing Answers: Usually, the SODB simply lists an answer under the question it answers. The answer can then be accessed by simple lookup. Answers can also be stored as raw data that is processed.

Data-Request: Any search for data initiated by a Requestor inputting a question. An infinite variety of searches can be done for data including searches that invoke functions to yield data. A data-request and a question can be considered synonyms. (By infinite searches we mean that there are infinite possible questions which can involve infinite different operations for finding or arriving at answers.)

Classifying a Data-Request: The SODB classifies data requests in order to diff rentiate between them and count them. However, as in any classification system, arbitrary rules must be established. SODB's classification of requests can therefore be infinitely variable.

Data-Use: When the SODB uses a piece of data as an answer or to arrive at an answer. Data-uses broadly fall into two types:

a) outputting the data as an answer or as part of an answer,

b) plugging the data into an algorithm that outputs the answer.

Classifying a Data-Use: As there are infinite algorithms and infinite types of answers, there are also infinite uses of data. The SODB has rules to classify these uses for the purpose of tallying the uses and paying royalties. For example the use of pi may be given a different royalty value than the use of the date of Lincoln's Birthday or the use of a passage from Shakespeare. As in classifying data-requests, there are no hard and fast rules.

Pay-off Meter (POM): The function that is the heart of the SODB. The POM has three sub-functions:

1) **The Demand Meter (D-Meter)** which tallies Data-Requests and Data-Uses over time to come up with an estimated **demand rate** for an answer (or for a piece of raw data),

2) **The Pay-off Formula (POF)** which takes the demand rate and calculates a **Pay-off Estimate (POE)** of the income a user will get for inputting the answer (or the data),

3) **Input Signal (I-Signal)** which outputs the POE and tells the answer (or the data) that may need inputting and, if necessary, instructions on how to input the answer (or the data).

The POM works most simply when the SODB's answers are listed under questions and the SODB can find the answers by simple lookup. For example, a Requestor may input the questions, "What is Lincoln's date of birth?." The SODB will do a lookup. Initially, the answer will not be in the SODB. The SODB will then store the question and tally each lookup. The SODB will also register the time of each lookup so that the rate of lookups over time will be known. The rate of lookups (the demand rate) for the answer will be fed into the POF to yield the POE. The I-Signal outputs this POE to every Requestor. Since answers are listed under questions, the I-Signal need not tell what answer need inputting nor how to input it. It is assumed that Requestors implicitly know that to enter an answer, they simply access the Supply Mode, enter the question and then enter the answer. Once the answer is inputted, the D-Meter still keeps track of the demand rate because the answer may be wrong. The POM is then still able to provide Requestors with the POE for correcting the answer.

When the SODB simply looks up answers, data-requests and data-uses can be measured by tallying lookups. The SODB can get more complicated though because the classification of data-requests and data-uses can get complicated. Furthermore, the demand rate can take into consideration prices of data as well. POM functions are discussed below, taking into account the issue of classifying data-requests and data-uses but not the issue of the pricing of data.

Demand Meter (D-Meter): The function that tallies data-requests and data-uses along with the time they take place in order to calculate the demand rate for a piece of data.

The D-Meter tallies:

a) data that is specifically searched for by name and not found. For example, a user may request a businesses phone number which is not in the SODB. This data-request can be tallied under the business' name;

b) data that is used but not specifically searched for by name. For example, a user may request the average price of airline tickets to Boston. Dozens of prices may be fed into an averaging function to answer this request. Each of these prices is used but has not been specifically requested by name.

c) data that is searched for by name and used (found). In these cases, the D-Meter only counts once. It does not count both the search and the find.

As discussed above, there can be an infinite variety of ways of classifying data-requests and data-uses, therefore, the D-Meter itself can be infinitely variable.

The key to the D-Meter is that it tallies data-requests and data-uses for data that satisfies two conditions: royalties are paid on the data and users can be instructed to input the data. The point of the D-Meter is to measure demand for specific data so that the demand rate can be fed into the POF which then yields the value of inputting that data. There would be no point in tallying requests for data that could not be named and therefore not be inputted by users.

The D-Meter does not necessarily measure the demand for all types of data that may be input into the SODB. For example, it is important to note that the D-Meter cannot measure the demand for potential answers. But, by measuring the demand for actual answers, the D-Meter can give users an idea of what the potential value is of inputting a potential answer. An example will illustrate both this situation and the issue of classification.

Assume the question inputted by a Requestor is, "What store has the lowest price on Sony Camcorder #1239?" Say there are 1,000 requests. Now it may be that ten stores have the same lowest price. What then is the demand for a given store? That depends on how the

SODB classifies the answer. The SODB may have a rule that only the first store with the lowest price can be outputted as the answer. This store becomes the answer and all royalties go to the inputter of this store. The SODB in effect turns the data-request into, "What stores have the lowest price and which among them was entered first?" Of course, the Requestor does not care which was entered first but the SODB may have default assumptions built into it to limit the size and number of answers outputted. Therefore, all other stores, even though they have the same lowest price are only potential answers (the first store may change its price so that another store takes its place). An inputter of a store with the lowest price does not know whether that input will generate any royalties or not. There is no established demand for that store.

On the other hand, the SODB can have a rule, for example, that all stores with the same price are equally part of the answer so the answer then has ten components. The demand rate for the store with the lowest price can then, for example, be divided by the number of components to arrive at a demand rate for each component (this calculation may actually be part of the POF). The classifications can be even more complicated. The second store inputted may be considered different from the first, location of the store may matter, and so on. The point is that the D-Meter tallies according to what data receives royalties and that depends on how answers are classified and that can be infinitely variable.

(Here in this continuation, we will often substitute the term "demand record" for Demand Meter because the key idea is that demand information is stored and then sent to the POF. There may be intermediate functions which calculate a demand rate (and these can be called part of a "demand meter") but that is not the main point. The same functions can be incorporated into the POF. It is easier to think of the parts of the Pay-off meter as being storage of demand information, calculation of a pay-off estimate, and outputting of the pay-off estimate. In the parent the issue may have been confused by having the demand meter do calculations. Because this application is a combination of the parent and an addition of new matter, we will use both terms. a future application will be stick to one way of looking at the steps of storing demand information and calculating demand.)

Pay-off Formula (POF): The function that calculates a **Pay-off Estimate (POE)**. The POF projects future demand for a piece of data based on the demand it has had in the past. Thus two variables are critical, N, the number of times the data has been requested and, T, the times those requests took place. Based on the rate of requests for a piece of data, the POF estimates how many future requests there will be and then multiplies that by a royalty rate to arrive at the POE. When a piece of data is already in the SODB, the POF uses the tally, N, of data-requests and data-uses as supplied by the D-Meter. (This point is elaborated on later in the text.)

Like any equation for a projection, the Pay-off formula can be infinitely diverse based on historical data and other factors. For example, the formula would include a historically based assumption of when demand would end. The POF may contain estimates not based on the actual demand rate for a specific piece of data but for pieces of data that are similar. Regardless of what historical assumptions are built into it, the POF is always a function of the demand rate. The values for N and T are plugged into the POF which has the royalty rate built in.

The royalty rate can, of course, be infinitely variable. There may be sliding scales for the royalty paid to an answer for example. And different data-requests and data-uses may have different royalty rates. (Technically, it is possible for the POF not to have a royalty rate and only calculate a projected request rate. In this case, the royalty rate would be known by users who could do their own calculations.) Also, the POF must have an arbitrary value for the POE when a piece of data has been requested zero times or one time. This value could be an amount or simply a message like, "POE unknown."

(Note: the SODB can include multiple Pay-off Formulas that apply to different types of data.)

I-Signal: The function that is the output part of the POM, the signal that tells Requestors what data is needed, what the value is of supplying it and how to supply it. When a Requestor requests an answer not in the SODB, the SODB outputs the POE. When a Requestor requests an answer that is in the SODB, the SODB outputs the answer and the POE for correcting, updating or improving it. (The POE may be outputted only upon request rather than automatically).

When the I-Signal outputs the POE it may also output the answer needed or the data needed. Usually, the answer needed is implicit from the question asked. If raw data is needed, the data needed may not be implicit from the question. In this case, if the SODB had the requisite functions for recognizing the data needed, the I-Signal might output the type of data needed as well. For example, "Need Answer to, 'What is the speed of light?', POE: $2." Finally, if necessary, the I-Signal could output instructions on how to input the data.

A basic SODB includes of the following elements and procedure as shown in figure 1:

SODB Elements

Computing means for executing SODB functions.

SODB Functions

a) inputting, storing, deleting and outputting data.

b)start mode

c) request mode

d) supply mode

e) lookup

f) registering charges

g) registering royalties

h) Pay-off Meter (POM)

SODB Procedure

Start Mode

1) User inputs user identification data, SODB stores it 1

2) User inputs supply or request command 1, SODB accesses appropriate mode 2.

Request Mode

1) Requestor inputs a Question 3, SODB

a)registers date-time of Request 4

b) executes a look-up 5.

2) If the SODB has the Answer, it

a)outputs the Answer 6

b)registers a payment due by the Requestor 7

c)registers a royalty due to the Supplier 7

d)adds one to the number of requests for that question 8, calculates the POF 9

e)outputs POE 10.

3) If the SODB does not have the answer it invokes the POM function which

a)checks if the Question is already stored in the Pay-off register 11 (has been asked before)

a1)if no, stores the Question 12, sets the number of requests for that question to one 13, calculates the POE using the POF 14

a2)if yes, adds one to the number of requests for that question 8, calculates the POE using the POF 9

b)outputs the POE to the Requestor 10.

Supply Mode

1) Supplier inputs a question 15, SODB executes a look-up 16 (this lookup is not counted as a data-request; only lookups in the Request Mode are so counted in the POM),

2) If the answer is not found, the supplier inputs the answer 17, SODB stores the answer to correspond with the question inputted and stores the Supplier ID data along with the answer 18, in order to credit the Supplier with a royalty each time the Answer is requested.

3) If the answer is found, the SODB outputs a message saying the answer is already in the SODB 19, if the Supplier intends to correct the existing answer, the Supplier inputs a command such as, CORRECT 20, and the SODB allows the new answer to replace the old answer 17 and allow new supplier ID data to replace the old ID data as well 18.

These elements and procedure are the heart of the SODB. The SODB would usually include other useful functions. Before detailing some of them, an embodiment of the basic SODB is described, a self-filling telephone directory (the SFTD). Then an embodiment is described which does more than lookup answers under questions.

1. The SFTD includes a list of names and corresponding telephone numbers, initially empty, a computer for storing the list and functions for inputting data into the list, outputting data from the list and looking up data in the list.

2. The SFTD also has a sign-on function that allows users to identify themselves for billing and payment purposes. The SFTD stores the ID data.

3. Users access the SFTD over the phone. The SFTD computer is equipped with phone-interface equipment. The SFTD accepts calls from two lines, a Request line and a Supply

Line. The Request line automatically puts users in the Request mode, while the Supply Line puts them in the Supply Mode.

4. Using the Request mode, a Requestor accesses the SFTD list by spelling a name over the phone into the SFTD's computer. Equipped with a speech recognition function, the SFTD recognizes the request and does a lookup. Equipped with a speech synthesizer, it then responds with a speech synthesized answer.

5. If the SFTD has a number corresponding to the name, it outputs the number and registers the charge due by the Requestor and the royalty due to the Supplier. One is added to the POM tally of data-requests, the time of the request is registered, and a new POE is calculated and outputted along with the number.

6. If not, the SFTD's POM is invoked and outputs a POE to the Requestor. The POM has several functions: a) it registers the time of the request, b) it checks if the request has already been stored in the POM register, c)if not, it sets the request tally to 1, stores the request and defaults the POE to the message "Insufficient Data to Estimate Pay-off," d) if the request is already stored, the POM advances the request tally by one and then calculates the POE using the POF (let us assume for illustration's sake, the POF divides the number of requests by the time period of those requests and then assumes that this rate will continue for 4 years. The formula then multiplies the number of requests over those four years by the royalty rate per request to arrive at the POE), e) outputs the POE.

7. A Supplier accesses the SFTD by spelling a name over the phone into the SFTD. The SFTD's speech recognition function recognizes the request and the SFTD does a lookup to see if a corresponding telephone number is already in the list. If the number is not in the SFTD, the SFTD allows the Supplier to enter the number and stores the Supplier ID data along with the number in order to properly credit royalties. If the number is already in the SFTD, the SFTD outputs a voice synthesized message, "Number is already in directory." If the number needs correcting, the Supplier then enters the command, CORRECT. The SFTD allows the Supplier to input the number using the SFTD's speech recognition function. The SFTD stores the number to correspond to the question, to the name that is, and also stores the Supplier's ID data with the number, in order to properly credit royalties.

Let us look at another embodiment, a lowest price locator, as shown in Figures 2a and 2b.

1. A lowest price locator (LPL) includes a list of product names and corresponding prices and stores, initially empty, a computer for storing the list and functions for inputting data into the list, outputting data from the list, looking up data in the list and processing data in the list.

2. The LPL also has a sign-on function 30 that allows users to identify themselves for billing and payment purposes. The LPL stores the ID data.

3. Users access the LPL over the phone. The LPL computer is equipped with phone-interface equipment. The LPL accepts calls from two lines, a Request Line and a Supply Line. The Request line puts users in the Request mode, the Supply Line puts them in the Supply Mode.

4. Using the Request mode, a Requestor accesses the LPL list by spelling a full product name over the phone into the LPL's computer. Equipped with a speech recognition function, the LPL recognizes the request 31 and checks 32 to see if the price is in its data-base.

5. The LPL registers 33 the time of the request.

6. If LPL finds no list of stores and prices under the product name, it checks 34 to see if the price has been requested before. If not, it stores the request 35, sets the request tally to one 36, calculates the POE 37 and outputs the POE 38. If so, it advances the tally 39, calculates the POE 40 and outputs the POE 38.

7. If LPL finds a list of stores and prices under the product name, it checks 41 for the lowest price. If it finds 42 more than one store has the same lowest price, it finds 43 the store whose lowest price was entered first and outputs 44 that store and its price. If not, it outputs 44 the single lowest priced store and its price. It then registers 45 the charge owed by the Requestor and the royalty owed the Supplier. It then advances the request tally 39, calculates the POE 40 and outputs the POE 38.

(Supply Mode)
8. A Supplier accesses LPL by the Supply Mode and spells a product name over the phone into the LPL. The LPL's speech recognition function 50 registers the name and allows the Supplier to input 51 a store and price

9. The LPL registers 52 the time of the inputting along with the store and price.

10. The LPL registers 53 the user's identification data along with the store, price and time.

11. The LPL checks 54 to see if there is list of stores and prices under that product name. If not, the LPL creates a list and stores the data in the list.

12. If the LPL already has such a list, the LPL checks 56 to see if the store inputted matches any store in the list. If not the store, price, time and user ID data are stored 57 in the list. If so, the data just inputted and registered is put in the list in place 58 of the data registered with the matching store.

13. The LPL finds 59 the lowest price.

14. The LPL checks 60 to see if there is more than one lowest price. If not, the single lowest price is held 61 for output. If so, the LPL finds the first, lowest price entered 62 and holds 61 it for output.

Some Comments on Data Processed in Lists and Tables

As the embodiment above indicates, the SODB is well suited to collecting data that is processed in what may generally be referred to as lists and tables. Let us then make a few comments about such data and how the SODB can be applied to collect it. Many kinds of answers can only be found by processing data in a list or table and often that data can only be collected efficiently by members of a community rather than a central authority. For example, usually the most efficient way for an economy to find the lowest price on a given item is through a system that allows people to feed in prices to a central list where the prices are sorted to find the lowest ones. This way is more efficient than having a central authority call all the sellers of the item in order to check prices. With a feed-in system, only the low price sellers need feed in. The SODB is, of course, a feed-in system.

We also note that, because of the nature of data tables, it is often advantageous for a single entry to include many sub-elements. For example, in a table of data on companies, each company entry might have many sub-elements, such as the name of the company, its telephone number, address, sales, number of employees, top officers and so forth. Thus a "single" entry can be used to answer multiple questions.

We also note that when a supplier enters data into a table, the SODB may suggest that the Supplier enter additional data as part of the entry. For instance, a Supplier might supply the answer to the question, "What is the telephone number of IBM?" The SODB might then also suggest that the supplier enter IBM's address. The SODB could guide the Supplier in how to enter extra data.

Additional Method For Outputting Pay-Off Estimates

We now describe an additional method the SODB can include for outputting a POE for data that answers or helps answer multiple questions. This method can be especially useful for data that is entered into and processed in lists or tables, for this type of data is often used to answer multiple questions.

As discussed in the Definitions section, the demand meter records multiple data-uses and sends this demand information to the POF which then calculates the POE for the relevant answer or "raw data". The Definitions section did not go in depth about how the Demand Meter registers multiple data-uses. It was understood that each time a piece of data was used, the use was recorded. Here we will elaborate on this topic and how multiple data-uses can lead to multiple POE's.

Before explaining how the SODB can output multiple POE's for an answer that is used for multiple questions, let us review how the SODB outputs a POE for an answer to a single question. A question is entered and before any answer is entered into the system, demand information is recorded and a POE is calculated for the question. After the corresponding answer is supplied, the demand information and POE can still be attached to the question or they can be attached to the answer, or they can be attached to both. It makes no difference concerning answers that answer one question.

But where an answer is used to answer multiple questions, then multiple POE's can be involved. Demand information can be stored for and a POE calculated for each question answered. In other words, the SODB keeps a demand record and corresponding POE for each question. The SODB can also keep a separate demand record for the answer itself. This record would include the demand records and POE's for all the questions involved.

For example, the answer to the question, "What was the lowest temperature at the South Pole yesterday?" might answer other questions, such as, "What was the lowest temperature on the surface of the earth yesterday?" Thus a demand record would be kept for both questions and a separate POE attached to both questions. The temperature data for the South Pole would also have its own demand record which could include the records of the two questions (and other questions the temperature data answers or helps answer). The POE for the South Pole data could then be calculated based on a combination of the demand information of the various questions involved.

The parent explained that the SODB registers multiple data uses, but did not illustrate multiple demand records. They are not illustrated here either, for they are easily implemented by those skilled in the art.

Now, when it comes time for the SODB to output a POE in response to a question, the POE can be the POE associated with the question or it can be the POE associated with the answer (based on demand information from all the relevant questions). The Pay-off Formula determines the POE to output.

But the new point made now is that human common sense will often be better than a machine rule (POF) for guessing the projected income for an answer that is used to answer multiple questions.

For example, the answer to the question, "What is the lowest price for a Walkman in the world?" might have as the answer, "$25 at Luskins." Another question that uses the same data for an answer might be, "What is the price for a Walkman at Luskins?" Now we will assume that the first question has a high POE, say $100, because we will assume that thousands of people want to know what the lowest price in the world is for a Walkman. And we will assume that the POE for the second question is low, say 10 cents, because we will assume that far fewer people ask the price for Walkmans at Luskins. So the same data, "$25 at Luskins," has a POE of $100 when it answers one question and 10 cents when it answers another.

Now assume that the price changes and the Luskin's price rises so that it is no longer the lowest in the world and thus no longer answers the first question. The remaining POE is only for the second question, 10 cents. And let us say that the new price is not in the system yet, and that a Supplier wants to enter the new price. But the new price may have an unusually high POE based on the "lowest-price-in-the-world" question it previously answered. Once the new price is in the system, the SODB will quickly register a drastic drop in demand for the Luskin's price but not until the new price is in. A person however can look at both questions that the data has answered and determine that the new POE will be low because the new price will only answer the second question (we assume the Supplier gets no extra reward for correcting the answer to the first question).

And so, a useful new feature not disclosed in the parent is that the SODB can include steps enabling Requestors to see more projected pay-off information than a single POE. The SODB can display some or all the information kept in the demand meter for an answer. Thus, the SODB can display:

a. all the questions that given data has answered or helped answer,

b. the actual royalties paid corresponding to each question,

c. the time periods the royalties were incurred and,

d. the current POE's for the questions.

If there is a large number of questions, their display can be ordered in some way, such as by showing the questions that have generated the most royalties or have the highest current POE's. By seeing the questions that the data has answered or helped answer, the Requestor can see whether supplying replacement data is worthwhile.

We note that the most common procedure might be for the SODB to first output the POE for the question that the Supplier is intentionally answering. The Supplier can usually signify which question by entering the question first and then the answer. In the case above, the Supplier would signify that he was answering the question, "What is the price of Walkmans at Luskins?" The SODB may then default to first showing the POE for this question, and then showing a combined POE and/or the POE's from other questions the data might answer.

(We also note that before an answer is in the system, the SODB can include means for anticipating multiple questions that an answer might satisfy. For example, a question might be, "What is the temperature in Moscow today?" The system might anticipate that the answer to this question can be used to answer other questions such as, "What is the average temperature in Moscow this month?" The POE would then reflect these anticipated data-uses. For the SODB to anticipate in this manner requires that the SODB have functions that identify comparable questions and answers and extrapolate demand information and POE's from them. Further. the SODB might have functions for identifying and registering that missing data could have been used to answer multiple questions. We do not describe such functions because they depend on highly specific situations. Users, of course, can make their own extrapolations.)

Additi nal Demand Information

We have shown above how the SODB gathers demand information on which to base a pay-off estimate. However, we showed only the most basic information being collected, the number of data-requests and the times of those requests. It often useful to gather more information. In fact, there is often a tradeoff between taking the time to gather more information versus the value of that information for calculating a more accurate projection of income, the pay-off estimate.

Formulas for projections have been known to contain thousands of variables. We will not discuss or describe nearly that number but we will note here a few more pieces of demand information that can be useful.

One piece of information that can be useful to register is whether a Requestor has asked the same question previously. In many cases repeat requests will mean misleading double counting of requests. For example, a Requestor might ask for the final score of a football game ten times before getting an answer (because the answer has not been entered until the time of the tenth request). It can therefore be useful for the SODB to include steps for registering whether a Requestor is making a repeat request.

In another variation of gathering such information, the SODB can ask the Requestor whether he has asked for the same answer before and whether the request new or is a "repeat." Asking the Requestor explicitly can be important in at least two cases. In one case the system may not record the Requestors of a given answer. In the other, the Requestor may know better than a machine based rule whether a request constitutes double counting or not. For example, a person might request an answer to the question, "What is the temperature of the ocean at Ocean City?" The answer can change rapidly. the Requestor will know if he is his request is a repeat or if he expects a different answer in each case. The point is that double counting depends on the situation and the user's common sense can be more valuable in identifying double counting than a machine rule.

Another piece of useful demand information that the SODB can register is how long they are interested in the answers the have requested. Many answers are only valuable for short periods of time. For example, the SODB might register dozens of requests for the score of a football game. From all these requests, the SODB might project a large POE. However, the SODB does not "know" that almost no one will be interested in the score shortly after the game in question is over. For the SODB to "realize" this fact, users must "tell" it. Thus, the SODB can ask Requestors to input a time period for which they are interested in an answer data. Even if the data is outputted, the SODB can still ask Requestors to input this information.

Furthermore, a Requestor could also specify how long thought others would be interested in an answer. Now, this opinion is a guess but can still provide valuable information to the SODB for calculating a projection of future demand. Taking our football score example, a Requestor can input that he is interested in the score of the game up until four o'clock. And he can say that he thinks demand for the score will taper off at eight o'clock. Say the game ends at 4'o clock. It is often better for the SODB to register and use such information than not.

In another variation of this type of demand information, the SODB might ask register users whether they want an answer sent to them and for how long the order stands. The Requestor would specify the time period that the answer could be sent until. The Requestor could also cancel the request. (We presume that the SODB in this case has the capability to send answers to E-mail boxes.)

Price Tests---Registering M re Demand Information

The parent did not delve into the issue of gathering price information from Requestors. It was assumed that Requestors knew the price of the answers they were requesting as, for instance, a person calling directory assistance or a 1-900 number would know the charges involved. The parent avoided price because the goal was to describe the core steps of an SODB. These steps include the gathering demand information about answers, feeding this information into a pay-off formula and outputting a pay-off estimate to Requestors. The key demand information the parent described (the number of times a question is asked and the times of those requests) is usually critical for making a projection of future income. Collecting this information did not, of course, preclude collecting other information about the demand for answers. Here we will describe steps for gathering information about what Requestors are willing to pay for answers.

Often it is neither practical nor desirable for Requestors to know the price of answers before the answers are requested. For example, the price of certain answers may change rapidly. The price of, say, today's weather report might be 20 cents while the price of yesterday's may be 2 cents. As another example, similar type answers may have very different prices. The phone number of the President's barber might cost 5 cents while the phone number of the President's private line might cost $5,000.

When the price of answers is not known in advance by Requestors, it is often useful to gather information on what Requestors are willing to pay for those answers because this information can be used by a POF to calculate the POE.

(We also note that gathering information on what Requestors are willing to pay for answers can be critical for setting the prices of those answers. We will discuss this issue in a future application.)

Now since the SODB can't read minds, it must perform what we will call price tests. These tests will not reveal exactly what people are willing to pay, but they seem to be the best that can be done. There are two fundamental price tests. One is where the system (seller) offers a price for an answer and the Requestor (buyer) can accept or reject the price. The other is where the Requestor offers a price and the system can accept it, reject it, or simply register it (if the answer is not in the system, the system usually need not accept or reject the offer but just record it in the demand meter and tell the Requestor that the answer is not in the system).

The world of commerce has evolved a great variety of price offers and counter offers that can occur in a sale situation. Earnest money can be pledged, time limits can be imposed, letters of intent can be written, discounts can be given, and so forth. Many of these price offers can have analogues in SODB price tests. Here we will describe mainly the basics, in which either the system makes an offer or the Requestor makes an offer. We will include some important additions but we realize that a great number of variations are possible.

The basic idea behind the system-offer price test is that the system can tally total requests along with the acceptance/rejection rate at a given price. This information is then fed into the POF. The resulting POE might then be correlated not only with acceptances but with total requests and with the acceptance/rejection rate.

The basic idea behind the Requestor-offer price test is simply that the system can register what each Requestor says he will pay. This information is also sent to the POF. The Requestor's offer is not necessarily just talk. If the answer is in the system, the Requestor would usually be charged the amount offered. Even when the answer is not in the system, the system can enable the Requestor to obligate himself to pay the amount offered provided the answer is entered into the system by a given time.

For price testing, it can matter if the answer is in the system or not. For example, take the answer to the question, "What time does *The Rockford Files* start tonight?" After this answer is in the system, the system might have a price assigned to the answer. But before the answer is in the system, the system might ask the Requestor to make an offer. Thus, different price tests can be used before and after the answer is in. Or, the same price test can be used in either case. There are four basic possibilities:

Before Answer Is In	After Answer Is In
System makes offer	System makes offer
System makes offer	Requestor makes offer
Requestor makes offer	System makes offer
Requestor makes offer	Requestor makes offer

It can also matter if the price test is done before telling the Requestor whether or not the answer is in the system. Thus we double the number of possible ways by which price tests can be done. We will not illustrate all the permutations but will illustrate enough to get the basic steps across, while recognizing that variations are possible.

The significance of the price tests in these different permutations is different and therefore the SODB can register information concerning whether the price test was done before or after the answer was in the system and whether or not the Requestor knew if the answer was in the system or not.

We will illustrate some price testing sequences. In the illustrations we will avoid repeating steps previously shown. For example, we gloss over the registering of data-requests and the times of those requests. The point is to show the new steps for executing price tests and using information from these tests in a POF for output as a POE. Also, we will assume that a given question has already been entered and that a demand record has been created for the question, as shown in figure 3. The rest of the figures, 3a-3d, show various price testing sequences. Also, in the figures, the term "question" will be used in place of "data-request" and "answer" will be used in place of "data." Also, when we say that the system registers a piece of information we mean that it stores the information in the demand record for the question being asked.

Finally, note that we also assume that the price offers that the system makes and the price thresholds that the system includes can be set in various ways: by the data-base manager, by the Supplier, by a price setting formula, or by some combination of these. We do not show the setting of prices but assume that step is done at the appropriate time in each case. Price setting will be addressed further in a future application.

Figure 3 shows the basic steps for registering the number of times a question is asked and the times of those requests. This demand information is stored in a demand record for the question and corresponding answer. Price test information is also stored in this demand record (in the parent this record was called the Demand Meter). As figure 3 shows, the SODB inputs 100 a question, then registers 101 the time of the request and checks 102 to see the question has been entered previously.

If the question is not found, the system creates 103 a demand record for the question and then stores 104 the time of the request and sets 104 the number of requests at 1.

If the question is found, the system stores 105 the time of the request and adds 105 one to the request tally.

The system then goes on to the steps for executing price tests. We will use the steps above as common preceding steps for four different price testing sequences. These different sequences are illustrated in figures 3a-3d. For the sake of concreteness, let us say that the question we will have in mind in all these illustrations is, "Who sells security holograms in the U.S.?"

In figure 3a a price testing sequence is illustrated in which the system presents 110 a price to the Requestor. The system enables 111 the Requestor to accept or reject the offer and the system does not tell the Requestor whether or not the answer is in the system.

If the Requestor rejects the price, the system registers 112 the rejection at that price, calculates 118 the POE, and outputs 119 the POE.

If the Requestor accepts the price, the system registers 113 the acceptance at that price, and then checks 114 to see if the answer is in the system. If the answer is not found, the system tells 115 the Requestor and then calculates and outputs the POE. If the answer is found, the system outputs 116 the answer, registers 117 the charge due to the Requestor and the royalty due to the Supplier and, calculates and outputs the POE.

Figure 3b shows a different price testing sequence where the system tells the Requestor whether or not the answer is in the system before the price tests. Further, the sequence has both price tests, one where the system makes an offer and the other where the Requestor makes an offer.

As shown in figure 3b, the system checks 120 if the answer is in the system. If the answer is in, the system tells 121 the Requestor and presents 122 a price. The system then enables 123 the Requestor to accept or reject the price. As before, if the Requestor rejects the price, the system registers 124 the rejection at that price and calculates and outputs the POE. And, as before, if the Requestor accepts the price, the system register 125 the acceptance at that price, outputs the answer, registers charges and royalties and calculates and outputs the POE.

Now, if the answer is not in the system, the system tells 126 the Requestor that answer is not in. The system then asks 127 the Requestor to make an offer. Here, as shown, the system can include steps for enabling the Requestor to make various offers.

The system can register 128 a non-binding offer. Here the Requestor expresses what he says he is willing to pay

The system can register 129 a binding offer to pay an amount up until a certain time. In this offer the Requestor not only states an amount he will pay but states a period of time his comitment is valid until. This type of offer can be *quite* important where certain kinds of answers are concerned. When binding commitments are involved, the Supplier can be fairly sure of getting a given amount of money for supplying a given answer. The system would also add to the POE based on Requestors who do not make binding commitments.

The system can register 130 binding offers that include a commitment of earnest money as a deposit.

(Also shown in figure 3b is a step 131 for registering the Requestor's opinion as to the reasonable price for the answer. This opinion is simply the Requestor's judgement and not a personal offer. The step is pictured because it can be important demand information in certain cases. The Requestor can have this option in other price sequences as well and can both make an offer and state an opinion.)

Once the system registers the Requestor's offer, the system, as usual, calculates and outputs the POE.

Figure 3c shows another price testing sequence that includes both types of price tests. In this sequence, the Requestor is not told before the price tests whether the answer is in the system or not. The main new feature here concerns the Requestor offer. The Requestor is asked to make an offer when the answer is in the system. In addition to asking for an offer, the system includes steps for registering when the Requestor makes an offer and steps for limiting the number of offers the Requestor can make.

If a Requestor can make unlimited offers when an answer is in the system, the Requestor will start low and keep going up. The Requestor will try to discover the system's threshold price ("bottom line"). Thus, the system needs to limit the number of offers the Requestor can make. This concern does not apply usually when the answer is not in the system because then the answer may have no price threshold attached to it.

(It is possible for a Requestor to have a confederate make an offer in an attempt to find the system's "bottom line." But with answers that cost a small amount this practice is unlikely or impractical. So, a feature limiting the number of offers a Requestor can make on an answer can be useful.)

The sequence in figure 3c limits the Requestor to one offer. (In figure 3d, steps limit the Requestor to one offer per a set period of time.)

In figure 3c then, the system checks 140 to see if the answer if found. If the answer is not in the system, the system presents 141 a price to the Requestor and enables the Requestor to accept or reject the price. The System then registers 142, 143 whether the price is accepted or rejected and tells 144 the Requestor that the answer is not in the system and then calculates and outputs a POE.

If the answer is in the system, the system then checks 145 whether the Requestor has made a previous offer. If yes, the system tells 146 the Requestor that he is ineligible to make an offer and then, as usual, the system calculates and outputs a POE.

If the Requestor has not made an offer, the system asks 147 the Requestor to make an offer. The system then registers 148 the offer. The system then accepts or rejects the offer. If the offer is rejected, the system tells 149 the Requestor that the offer is rejected and registers 150 that the Requestor has made an offer for this answer. Then, as usual, the system calculates and outputs a POE. If the offer is accepted, the system outputs 151 the answer, registers the charges and royalties due, and calculates and outputs the POE.

Figure 3d shows a price testing sequence in which only the Requestor makes offers. Here steps are shown that limit the Requestor to making one offer per time period. The point, as mentioned previously, is to limit the number of offers that a Requestor can make in order to get the Requestor to make a higher offer. We note in figure 3d that if a Requestor makes an offer before the answer is in the system then this offer is not subject to a time period prohibition. The Requestor is free to make a different offer once the answer is in.

So as figure 3d shows, the system checks 160 whether the Requestor has made an offer that has been rejected.

If yes, the system checks 161 to see if the pre-determined time period has expired. If not, the system tells 162 the Requestor that he cannot make another offer and, as usual, calculates and outputs a POE.

If the time period has expired, the system asks 163 the Requestor to make an offer. The system registers 164 the offer. The system then checks 165 to see if the answer is in the system.

Likewise, if the Requestor has never made an offer before that has been rejected, the system ask for an offer, registers the offer and checks to see if the answer is the system.

If the answer is not in the system, the system tells 166 the Requestor that the answer is not found and, as usual, calculates and outputs a POE.

If the answer is in the system, the system checks 167 the price threshold and accepts or rejects the offer. If the system rejects the offer, it tells 168 the Requestor that the offer is rejected and sets 169 a time period for when the Requestor can make another offer for the answer, and, as usual, calculates and outputs a POE.

If the system accepts the offer, it outputs the answer and registers charges and royalties and calculates and outputs a POE.

Brief Note About Price Tests With Price Ranges

Normally a price offer is at a single price. However, the SODB and Requestors can each present offers as ranges, especially when an the answer requested is not yet in the system. For example, marketing research polls that ask people what they are willing to pay for an item often ask in terms of price ranges. The point is that information about price ranges can be more useful than single prices (we also note the important point that Requestors, and the SODB, can offer different prices corresponding to different times.)

There is another reason though that the SODB can present offers in ranges and that is because the nature of the SODB is such that a user may indeed end up paying a price that is in a range. Here we have the fundamental idea of projected price.

The SODB may present a Requestor with a projected price. For example, the SODB might present an offer whereby the price of an answer is between 20 cents and $2.00, with the projected or expected price being 50 cents. Taking our hologram example, a Supplier who does research and compiles a list of hologram producers might want to be rather sure of being compensated for his time in compiling the list and entering it into the system. He might want, say, $20. And so, the SODB might set the initial price for the hologram answer high, in order to better insure that the Supplier will be paid the $20. Thus the first ten Requestor might be charged $2 each. However, say another 100 Requestors ask for the same hologram answer. These can be charged less, say 20 cents each, and the original Requestors can be

rebated an amount. Thus the actual price the original Requestors pay is not definite, but a projected or expected price.
Just as the system calculates a POE, it can calculate a projected price.

We will not delve into this idea further here but note that in certain situations the principle of projected price, like projected income, is not only fundamental but highly useful in getting answers into the SODB.

Brief Comments on the Types of Answers Suppliers Can Enter

In general, we get answers from computers in three ways. One is by straight look-up, direct correspondence. For example, if someone asks, "What type of wood are Stradivarius violins made out of?" the answer could be stored in a computer under the question.

Another type of knowledge is what might be called "algorithmic" in the sense that information is compressed in an algorithm. For example, one could ask, "What the length of the hypotenuse of a right triangle with sides 3 inches and 4 inches long?" This answer could be stored to correspond to the question. One could make a huge look-up table with answers to questions about the lengths of different hypotenuses. A more efficient way of representing this information though is by the well known Pythagorean Theorem. This theorem allows you simply to plug in the relevant numbers and let the computer calculate an answer.

The SODB can be adapted to calculate answers from algorithms. For example, if 1,000 people ask questions about the length of the hypotenuses of different triangles, a user might realize that, rather than answer each of these questions, she could enter the theorem and enable people to plug in the appropriate numbers. The user that entered the formula and the interface allowing people to enter the numbers could then get the royalties for the questions the theorem answers.

It is thus possible to enable users to enter formulas and an interface procedures for Requestors to be able to use the formulas to get desired answers. The problem is that the SODB cannot come up with a POE for the algorithm because doing so requires the intelligence to realize that a group of different questions can be answered with a single algorithm.

Likewise with tables of data that is processed; it is possible to enable Suppliers to enter sets of functions that create tables such that data can be entered into the tables and then be processed. Again, the SODB does not have the intelligence to assign a POE to such tables + functions. As with algorithmic answers, we note the possibility of enabling Suppliers to enter tables + functions and paying Suppliers for the usage of the tables. While having such features can be quite important, it is not he main point of the system.

(In the parent we presumed that the tables and functions can be part of the SODB. After all, there is nothing new in processing data in tables. What is new in the SODB is how economic values are attached to given data (whether that data is in tables or not) to induce users to supply that data.)

Brief Note on the Form of the Invention

Before going on to other features of the invention, it is worthwhile to pause and discuss the form of the invention. Because it is to be used by a community of people in different locations, the invention comprises a network in which terminals in various locations are used to input data requests and supply data. The terminals can be anything from telephones to super computers.

The data itself can be stored centrally or in nodes throughout the network. For example, certain users might request the full text of *Dracula*. Other users might want the film version of *Dracula*. And all this data can be stored centrally. Or the text of *Dracula*, the book, might be stored in a computer owned by, say, the Library of Congress, while the film data might be stored in a computer owned by, say, a film studio. Because of the added communications costs, highly decentralized storage of data is not usually the most efficient method where the SODB is concerned. Nevertheless, real world concerns might dictate such decentralization. A movie studio, for example, might not want to put its copyrighted movie in someone else's computer for distribution to the public.

(One problem in discussing the issue of centralized storage is that the very concept of centralized storage is blurry in this age of sprawling networks. We will not try to define the notion crisply here, but will rely on peoples' intuitive notions.)

While the storage of data itself may be decentralized, the gathering of demand information about data-requests (and the calculation and outputting of POE's) must, in general, be highly centralized. For example, say we have a data-request, "How many paintings are in the Louvre?," and say that a dozen users request the answer to this question. It does no good if the twelve data-requests are all registered on different systems. There needs to be a central tally showing that there have been twelve requests for the data. That way the POE corresponds to the demand of twelve people. Otherwise the POE in each system would only correspond to one request.

In fact, the goal of the SODB is to collect the demand for given data centrally. That way the pay-off for supplying the data is higher and the data can cost less per user. Moreover, the more demand for a piece of data, the more likely the data will be entered into the system. If the demand is decentralized then there is no way to accumulate the demand and that defeats the purpose of the system. Of course it is conceivable that the demand could be registered throughout the system but it would still have to be tallied, matched up, somewhere to yield a total figure which would then lead to the maximal POE.

(The economic efficiency of accumulating demand information does not mean that a single SODB will store all the world's data-requests. An SODB is meant to be used by a community and a community can be defined narrowly. For example, a company might have an SODB for its employees. Data-request demand information would be stored centrally though and not in every employee's computer.)

We have mentioned that the data itself might be stored in a decentralized manner. However, if the SODB does not store data centrally, it must at least store *pointers* to the data centrally. For example, if a Requestor asks for a given piece of data, the SODB must be able to tell if the data is in the system or not. To do this, a pointer would identify whether the data was in and where it was located. Thus a data-pointer is surrogate for the data itself. In the case of decentralized storage of data then, a Supplier who enters data into the system would have to enter a pointer centrally while entering the data into a given storage computer.

It is also possible that the SODB only outputs routing information to the Requestor but does not make the connection to the storage computer. In this case, the SODB is really a new kind of signaling mechanism that tells users where data is stored and tells users the potential pay-off of storing and selling data. This form of the invention was not envisioned in the parent and is noted here.

Another aspect of the SODB that can be decentralized is paying royalties and collecting charges. This can be done at the nodes where the data is stored. Again, this form of the invention was not envisioned in the parent but is noted here. However, even if payments are transacted in a decentralized manner, payment data would still be sent to the central SODB location because such data is usually important demand information to be used in calculating pay-off estimates.

Brief Note on the Importance of the Flexibility f the Royalty Rules

One of the advantages of the SODB is that the royalty rules and the POF are infinitely variable. Thus, the system Manager can adjust the formula to reward certain actions such as the correcting of answers. We will go into the importance more in a future application but here we will note one of the most important consequences, starting up the system and attaining critical mass.

For many types of fee based data-base systems, the problem is starting up and gathering enough initial data and enough initial customers. This problem is often referred to as the critical mass problem. The idea is that if enough users use the system the system will be profitable and self-perpetuating. But it is a chicken and egg problem, for often no users can be gotten until the data is in the system.

The beauty of the SODB is that it enables the System Manager to provide incentives that can jump start the system. For example, if your plan is to start a lowest price locating system, a huge obstacle is how to convince thousands of sellers nationwide to feed in their prices so that the prices can be sorted. We met a similar problem in the very beginning when we discussed the problem of even keeping a data-base of telephone numbers up to date. The problems with a lowest price locator are worse.

If we agree though that people would be interested in lowest prices we can see that if the system got started it might be self-perpetuating for buyers would want to check lowest prices and the low price sellers would, out of self interest, want to display their prices. So let us assume that once the system got going it would have value for users.

In order to jump start the system the Manager can adjust the royalty rules so that the people who are the first to enter the lowest price of a given item get a share of future income from all the lowest prices, for that item for a period of time. For example, say that the item is a Sony Walkman (we'll pretend there is just one model in the world). Then the royalty rules can be set such that the person who enters the lowest price will get a share of the royalties of all subsequent lowest prices, for a period of, say, 5 years. Now, if there is no price in the data-base then the first person to enter the price is the lowest. That is not a reasonable way to get the system going. Therefore, the System Manager can set a rule such that the "first" lowest price Supplier is considered to be the person who has entered the lowest price that is valid at a given date and time.

The System Manager can set the royalty rules such that, for instance, the Supplier of the lowest price for a Walkman on December 24th, at noon, gets a small share of the royalties for all lowest prices entered for the next 5 years on a Walkman. The reward might be, say, $200. Thus the System Manager can set up a competition to be the lowest price Supplier on a given date and time. The competition might last, say a couple of months. At the end of this competition, a truly low price might be entered and the system may be off an running for that item.

We are using this illustration just as a representative example of the advantages of being able to adjust the POF (the royalty rules really).

There are many other advantages of being able to adjust the royalty rate and thus the resulting POE's, but this use above is the inventor's favorite.

Additional Functions

An SODB should include more functions than the basic ones described above. Some useful functions are described below.

Matching Functions

The SODB is a matching machine in two senses, both critical. First, it matches questions (data-requests) and keeps a tally of how many times the same, matching questions have been asked. Second, it matches answers to questions. In both types of matching, problems can arise due to the nature of language and the nature of questions and answers themselves. Therefore, the SODB should have functions to increase the chance of accurate matching. Examples of such approaches are best match algorithms.

a) Infinite Ways to Ask the Same Question

There are multiple, in fact infinite, ways to ask the same question. Two questions that have the same meaning may not be matchable because they have a different form. And so, the goal of the SODB is to try to make questions with the same meaning take on the same form. The SODB can therefore have a function that takes a Requestor through a standard input structure so that Requestors have a better chance of posing Questions in matching forms when the questions have the same meaning. This structure is easiest with simple questions such as, "What is the telephone number of John Smith?" A Requestor might simply input the name "John Smith," which would of course, match other inputs of "John Smith." This example brings us to the next problem.

b) Questions Can Have Multiple, and Possibly Infinite, Answers

To match an answer to a question, there needs to be a single answer. However, as discussed previously, the phrase "one answer" is not very clear. An answer can have multiple components. For example, the question, "What is John's telephone number?" might have an answer that includes multiple numbers because John might have several numbers. However, the answer could still be considered a single answer because the numbers are that person's numbers, which is what the requestor asked for. The Requestor though would not want multiple numbers of various people with the same name. For example, a person entering "John Smith" looking for a number might find hundreds of numbers. In fact, most all questions can have multiple answers, even seemingly specific questions such as, "What is John's weight in pounds?" The answer may be 150 or 150.11 or 150.1111, and then any figure depends on when he was weighed, with what scale, and so on. To narrow answers down, we use implicit default assumptions, some of which we build into our data-bases. In addition to these assumptions, the key to narrowing possible answers is to specify enough information in a question to make it highly likely that only one answer will be given. Specifiers such as full name, location, ID number, time, source of information and so on can often narrow the possible answers to one. The SODB can include a function that asks the Requestor to pose the question more specifically. The SODB can also include a function that picks one answer out of a set of equivalent answers. For example, the answer to the question, "Where is the lowest price on a certain compact disc?" might be many places. The SODB might pick one at random.

Quality Control Functions

Quality control of answers in the SODB is essential. The SODB can have many functions to provide incentives and sanctions that encourage Suppliers to provide accurate answers. A general incentive is that a corrected answer will displace a wrong answer and garner royalties. The SODB may have rules to define what a wrong answer is but these rules cannot cover all situations. Disputes may arise as to whether answers are accurate and these dispute may have to be settled outside the system by the Manager of the SODB. Some quality control functions are listed below.

a) The SODB can have a function that stores identification information about an Answer such as the time it was supplied and the primary source (the primary source and the Supplier may or may not be one and the same).

b) The SODB can have a function that allows users to input a claim that an answer is wrong and send that claim to the Manager.

c) A user, Requestor or Supplier, can claim that an answer was intentionally supplied wrongly. The SODB can have a function that allows a user to record this claim and send it to the Manager.

d) The SODB can have a function that allows a user to request that a manager inspect an answer. The function can also register a charge for this inspection.

e) The SODB can have a function that allows the Manager to register that an answer is wrong and to register that wrong to a Supplier.

f) The SODB can have function that keeps a record of the wrong answers a Supplier has provided. This function can also disqualify a Supplier who has inputted too many wrong answers.

g) The SODB can have a function that charges Suppliers an amount of money, a penalty, for providing wrong answers.

h) The SODB can have a function that rewards a user who discovers that an answer is wrong. Such a function can charge a penalty to the Supplier of the wrong answer and pay the penalty fee to the discoverer of the incorrect answer.

i) The SODB can have a function that pays Suppliers to update answers. Let us call such a Supplier an Updater. For example, a price that was originally entered correctly might become outdated. A user who discovers this can be paid for changing the answer to a correct one. The user would be paid royalties that the new, correct answer would generate. However, sometimes, when an answer is changed, it may receive no royalties. This is particularly true with prices and other time sensitive offers. For example, the answer to the question "Who sells HP printers for the lowest price?" might change. The Updater might find out that the current answer in the SODB is wrong. But the Updater might not be able to Supply the correct answer. That may have been supplied by someone else. In these cases, the SODB can have a function that pays the Updater a share of the Royalties owed to the Supplier of the new answer and/or of the old answer. Or the SODB may be able to credit the Updater in other ways, such as crediting him with free answers.

j) To cheat, a person might have a confederate change an accurate answer to a wrong one. The person would then re-enter the answer correctly and claim royalties. The SODB can have a function such that if an answer reverts to a previous answer within a given period of time, royalties will be paid to the Supplier of the previous answer, provided the previous answer was accurate. With static facts, such as a person's birthday or the speed of light, the first person to supply the answer accurately would usually claim all royalties. With changing facts, such as prices, the time allowed for reversion could vary depending on the situation.

k) The SODB can have a function that "confirms" answers by making sure that they are outputted to Requestors only after having been inputted by more than one Supplier.

l) The SODB can have a function that allows Suppliers to enter answers only after having entered a passcode.

m) If accessed by voice, the SODB can have a function that records the Supplier's voice for identification.

n) The SODB can have a function that audio records the Supplier receiving an answer from the primary source of that answer. For example, a Supplier could be getting a price from a store. In order to insure that the store cannot renege on this price, the Supplier might want to record the conversation. Thus the SODB can have a call forwarding function in which the Supplier calls through the SODB, the SODB connects the Supplier to the store and then also records the conversation. To reduce recording costs, the recording might be done randomly.

Deleting Data Function

The SODB can have a function to get rid of "deadwood" by deleting answers, raw data, data-requests and data-uses whose demand rate drops too low. For example, the SODB can automatically delete any answer that has not been requested or any question that has not been asked for a period of time.

User Fee Functions

To offset costs and to encourage efficient use, the SODB can have a function that charges users for connect time, for the storage of answers and for any other usage of the data-base.

Pay-off Meter Functions

a) A user might prefer not to have the POE outputted automatically but instead upon request. Therefore, the POM can have a function that allows Requestors to request a POE.

b) A function can be added to the POM that tells not only the Pay-off Estimate but also an estimated per hour rate. Thus the Pay-off Formula would have to include an estimate of the time it takes to input the necessary data. From this estimate, a per hour estimate follows.

c) The Pay-off Formula can calculate a second POE, one that is a percentage of the original POE and could be called a Referral Fee. This fee would be due a person, a Referrer, who alerted a Supplier to enter an answer. This function would allow a Supplier to input the name of the Referrer. The function would then credit royalties to both the Supplier and the Referrer. These two would normally share the original royalty amount.

d) The Pay-off Formula can calculate the Pay-off per component in an answer. There are infinite ways to assign a value per component. The Pay-off Formula could, for example, simply take the POE and divide it by the number of components, x, in an answer. The SODB would also have a function that tallies the components.

Requestor/Supplier Functions

a) A Requestor may not want to supply certain data because another Requestor might beat him to the punch. Therefore, the SODB can have a function that reserves the right to input the data. The Requestor could enter a command, such as RESERVE, after hearing the POE for the data. The function would store the Requestor's ID data along with the Requestor's question. Then, for a period of time, the SODB would allow only that user to enter the necessary data. This function would also alert other users that the data was reserved for that time.

b) A Requestor who becomes a Supplier may not want to bother re-entering a Question that he previously asked when in the Request mode. The SODB can then have a function whereby this user, when in the Request mode, could enter a command, such as "WILL SUPPLY", after hearing the POE for the answer. The function would store the Requestor's ID data along with the Question. Then, when the user is in the Supply mode, the function would, upon a command, such as PREVIOUS, look-up the last question that the user had asked. The data inputted by the user would then be stored to correspond to that Question.

c) A user who intends to be a Requestor might enter the Supply Mode, using that mode to check whether an answer is present in the SODB. (A user can check whether an answer is present using the Request or Supply Mode.) If the answer is present, the SODB can have a function that allows the user to automatically switch modes upon a single command and have the answer automatically outputted and a charge registered to the user. This function may seem trivial but an important issue lies behind it. The SODB is a feedback system different from other data-bases in that it forms a tight feedback loop based on royalty incentives provided to users who normally would pay to receive data. With certain data-bases, suppliers, who do not pay for receiving data, may be able to check on the potential royalty revenue from a piece of data. But, for the first time, with an SODB, this pay-off information is made directly available to users who are seeking data and who are charged for it if it is in the data-base. This fact makes a great difference for it creates a tight, efficient feedback system that is new.

Probabilistic Payment Functions

If payments and royalties for data are quite small, it is very advantageous to use an expected value payment method (EVPM). An EVPM is described in patent no. 5,085,435 and 5,269,521. Please see this patent for an explanation of the method.

The main question in an EVPM is how to insure fair bets. In this case the bets are between the SODB and its users, both Requestors who owe money and Suppliers who are owed money. We will take the case of Requestors who owe money. The principles involved extend to Suppliers. Cheat-prevention methods are described in the patents above. Two examples of cheat prevention methods that can be applied to the SODB are given below.

Numbers Game Method: In the illegal Numbers Game, results were often determined by one number, for example the last three digits of the handle at the track. Anyone who picked that number would win. Thus, one number decided thousands of bets. Likewise the SODB's Payments Register can set up EVPM bets with each Requestor. Charges registered one day can all be decided by the daily lotto number the next day. For example, assume the stakes are set at $10.00. The bet then is decided by the last three numbers in the daily lotto. (See EVPM patent). So, the Payments Register register the charges owed by all Requestors during one day. Then the next day, the daily lotto number is announced. The Payments Register takes this number and applies it to every bet is made with Requestors on the previous day.

The only problem with such a method is that it can truly be feast or famine. For example, assume all charges one day are 10 cents. The SODB only has a 1% chance of winning bets if the stakes are $10. Therefore, the SODB stands a 99% chance of getting nothing and a 1% chance of getting 100 times its money. In order to even out the income stream, the Payments

Register might assign to each Requestor an extra number to be added to the lotto number. The extra number might be part of the Requestors ID number, for example. These extra numbers would be random or nearly so in order to even out the wins and losses from bets. As long the extra numbers are agreed upon by Requestors before the lotto number is announced, all is fair.

Probabilistic Metering Method: Normally, when people use an on-line data-base or phone system or any usage sensitive system, there is a metering component that measures usage and ultimately determines charges. The SODB has that with its Payment's Register. However, registering charges and then billing for them can be a large cost. Therefore, it would be advantageous to do the metering on a probabilistic basis by EVPM. For example, the meter might be off 90 percent of the time but, when on, the charges applied would be at 10 times the normal rate. The periods of time the meter is on and off are determined by a random number supplier that picks a number, in this case an integer from 1-10.

The SODB's Payment's Register can have a Probabilistic Metering Function (PMF) that randomly determines the time periods during which the SODB will register charges to Requestors (and register royalties to Suppliers). The function is described below.

1. A period of time is broken up into sub-periods. For example, a day might be broken up into minutes.

2. The probability that the meter will be on during a sub-period is decided upon by the SODB manager.

3. Each sub-period has assigned to it a random number that determines whether the meter will be on or off during that period. The number is chosen by a random number generator such that the probability of the meter being on is the probability that the SODB manager has decided on. With each sub-period having a random number chosen, a sequence or list of "on's" and "off's" is created. This list is inputted into the PMF.

(The list is supplied by an independent source that generates the numbers. The independence of the source is necessary to verify whether or not the SODB's sequence if fair.)

5. The PMF has a clock and a sub-function that, upon the clock's arriving at each sub-period, checks the list to determine whether the meter should be on or off. The sub-function turns the meter on and off as determined by the on/off list.

6. The clock is synchronized to an independent clock so that fairness can be assured.

7. When the meter is on, the Payment Register registers charges and multiplies them by the inverse of the probability that the meter would be on. Thus, if the meter is to be on 1/10 of the time, the charges would be 10 times normal.

Probabilistic metering by this method offers an efficient way to insure fair bets and also a way to smooth out the wins and losses from bets. Perhaps more importantly, it allows Requestors not to have to input the ID data unless they lose bets. There is no reason to input one's identity if one does not have to pay. Thus the inputting of ID data is eliminated from the Start mode. This can be a very advantageous for people in a hurry. It means they only have to identify themselves for billing purposes when they lose the bets. Of course, people might not pay if they haven't identified themselves. However, in addition to honor, it is possible in some cases to gather evidence to trace Requestors. It is possible to capture the Requestor's voice, for instance, if the SODB is accessed by voice. If the SODB is accessed by computer, the computer may be traced.

Claims

I claim:

1. a data collection and retrieval system comprising in combination the following elements and steps:

a computer and data-base having:

--input means for inputting data-requests and data corresponding to said data-requests, along with user identification information,

--output means for outputting said corresponding data, along with projected pay-off estimates,

--memory means for storing said data-requests and corresponding data,

--processing means for comparing data-requests, finding corresponding data, registering times of inputs, calculating formulas and registering charges and payments due to users,

said computer performing the following steps upon the inputting of a data-request by a user:

a. registering said user's identification information in said data-base,

b. registering user's preference in supplying or retrieving the data corresponding to said data-request,

c. if the user prefers to supply said corresponding data, looks for said corresponding data is said corresponding data-base,

c1. if corresponding data is found, outputting a message telling the user that the data is already in the data-base,

c2. if no corresponding data is found, allowing the user to input the data, storing the corresponding data and registering that royalties are due to the user when said data is requested,

d. if the user prefers to retrieve data corresponding to said data-request,

d1. registering time and date said data-request is inputted,

d2. registering amount user is willing to pay for data corresponding to said data-request,

d3. finding data corresponding to said data-request is in the data-base,

d3a. if corresponding data is in the data-base, outputting the data, registering a charge due by said user who inputted the data-request and a royalty due to the user who supplied the data, adding one to the number of said data-requests, calculating a pay-off formula that projects the estimated royalties due to a user who inputs the correct data corresponding to said data-request, and outputting the resulting pay-off estimate,

d3b. if no corresponding data is in the data-base, checking if said data-request is stored in the data-base,

d3b1. if no, storing the data-request and setting the number of said data-requests to one, and calculating said pay-off formula,

d3b2. if yes, adds one to the number of said data-requests, and calculates said pay-off formula,

d3b3. outputting the resulting pay-off estimate.

FIGURE 1

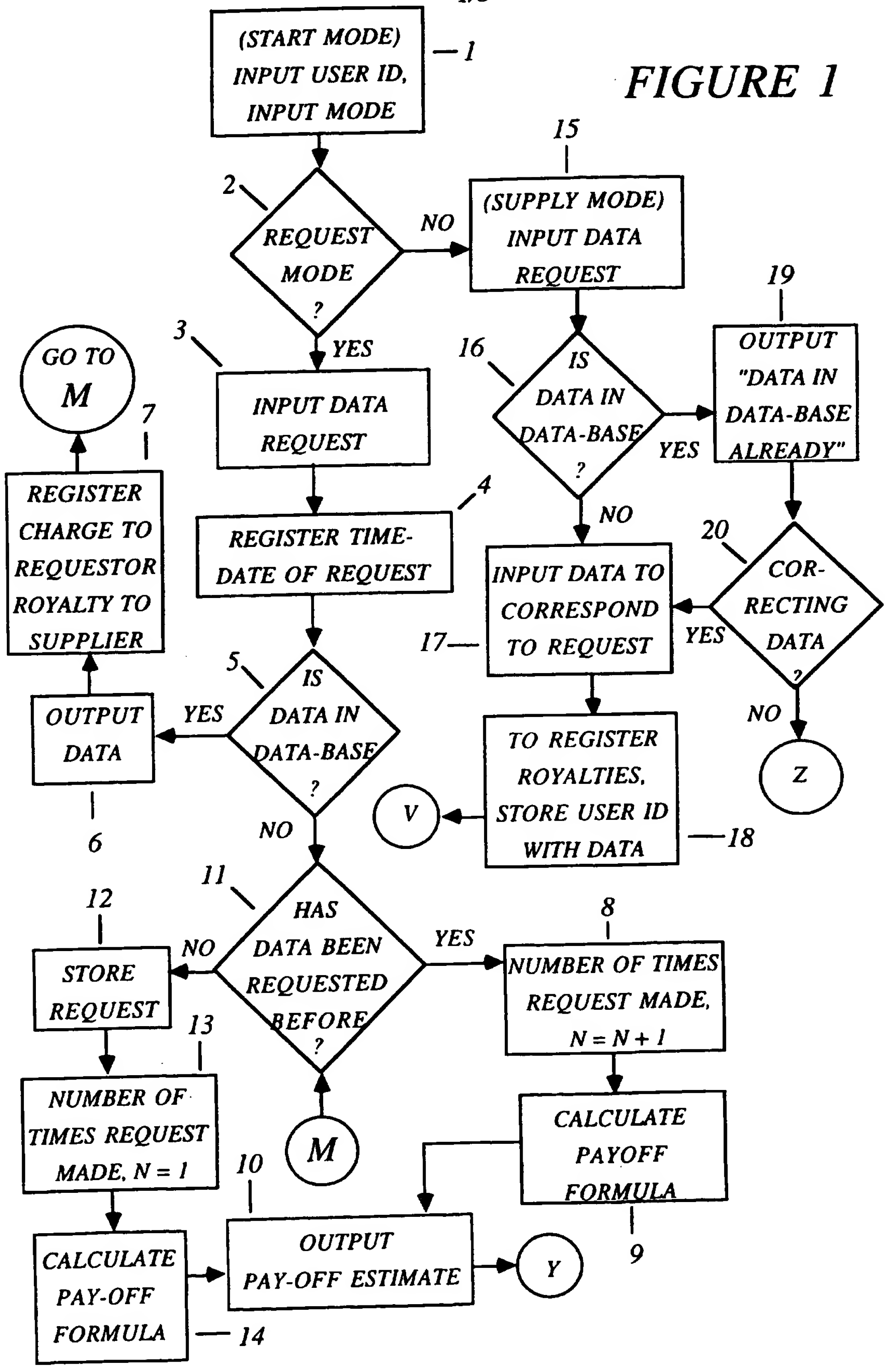
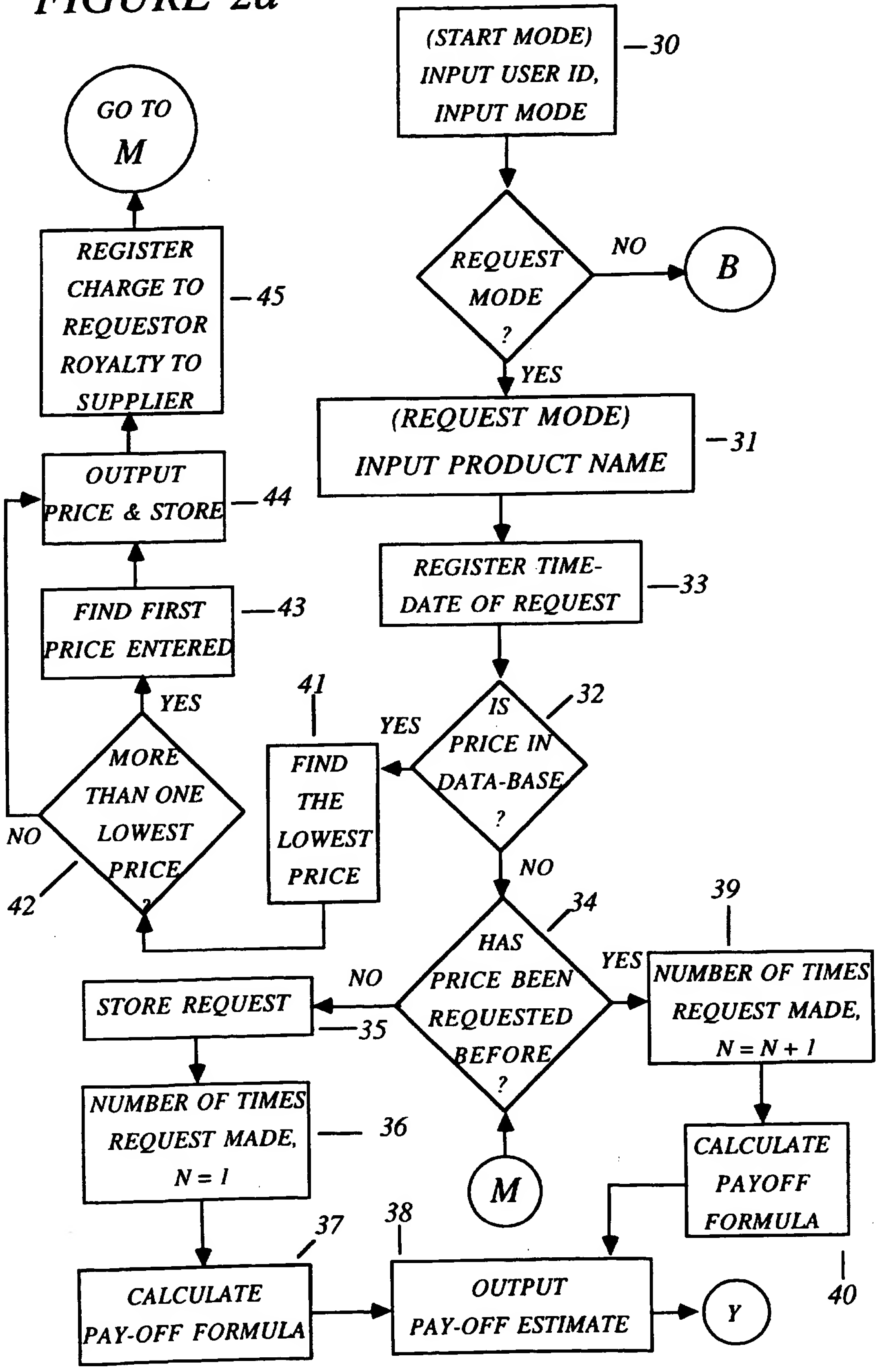

(START MODE) INPUT USER ID, INPUT MODE
1
2
REQUEST MODE ?
NO
YES
15
(SUPPLY MODE) INPUT DATA REQUEST
3
INPUT DATA REQUEST
4
REGISTER TIME-DATE OF REQUEST
5
IS DATA IN DATA-BASE ?
YES
OUTPUT DATA
6
REGISTER CHARGE TO REQUESTOR ROYALTY TO SUPPLIER
7
GO TO M
NO
11
HAS DATA BEEN REQUESTED BEFORE ?
NO
12
STORE REQUEST
13
NUMBER OF TIMES REQUEST MADE, N = 1
CALCULATE PAY-OFF FORMULA
14
M
YES
8
NUMBER OF TIMES REQUEST MADE, N = N + 1
CALCULATE PAYOFF FORMULA
9
10
OUTPUT PAY-OFF ESTIMATE
Y
16
IS DATA IN DATA-BASE ?
YES
19
OUTPUT "DATA IN DATA-BASE ALREADY"
20
CORRECTING DATA ?
YES
NO
Z
NO
INPUT DATA TO CORRESPOND TO REQUEST
17
TO REGISTER ROYALTIES, STORE USER ID WITH DATA
18
V

FIGURE 2a


(START MODE) INPUT USER ID, INPUT MODE
30
REQUEST MODE ?
NO
B
YES
(REQUEST MODE) INPUT PRODUCT NAME
31
REGISTER TIME-DATE OF REQUEST
33
IS PRICE IN DATA-BASE ?
32
YES
FIND THE LOWEST PRICE
41
MORE THAN ONE LOWEST PRICE ?
42
NO
YES
FIND FIRST PRICE ENTERED
43
OUTPUT PRICE & STORE
44
REGISTER CHARGE TO REQUESTOR ROYALTY TO SUPPLIER
45
GO TO M
NO
HAS PRICE BEEN REQUESTED BEFORE ?
34
NO
STORE REQUEST
35
NUMBER OF TIMES REQUEST MADE, N = 1
36
CALCULATE PAY-OFF FORMULA
37
YES
NUMBER OF TIMES REQUEST MADE, N = N + 1
39
CALCULATE PAYOFF FORMULA
40
M
OUTPUT PAY-OFF ESTIMATE
38
Y

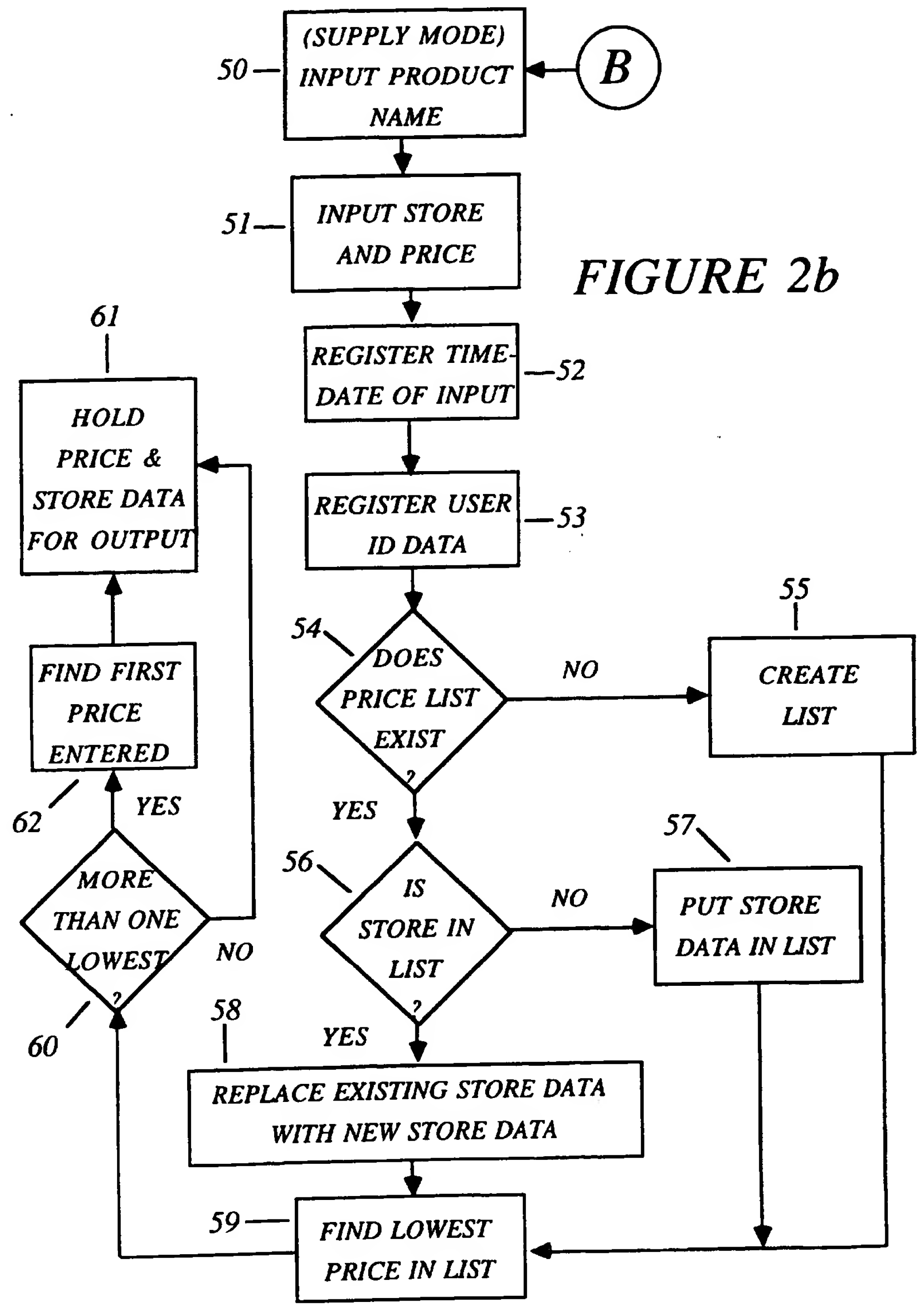

(SUPPLY MODE) INPUT PRODUCT NAME
50
B
INPUT STORE AND PRICE
51
REGISTER TIME-DATE OF INPUT
52
REGISTER USER ID DATA
53
DOES PRICE LIST EXIST ?
54
NO
CREATE LIST
55
YES
IS STORE IN LIST ?
56
NO
PUT STORE DATA IN LIST
57
YES
REPLACE EXISTING STORE DATA WITH NEW STORE DATA
58
FIND LOWEST PRICE IN LIST
59
MORE THAN ONE LOWEST ?
60
NO
YES
FIND FIRST PRICE ENTERED
62
HOLD PRICE & STORE DATA FOR OUTPUT
61

FIGURE 2b

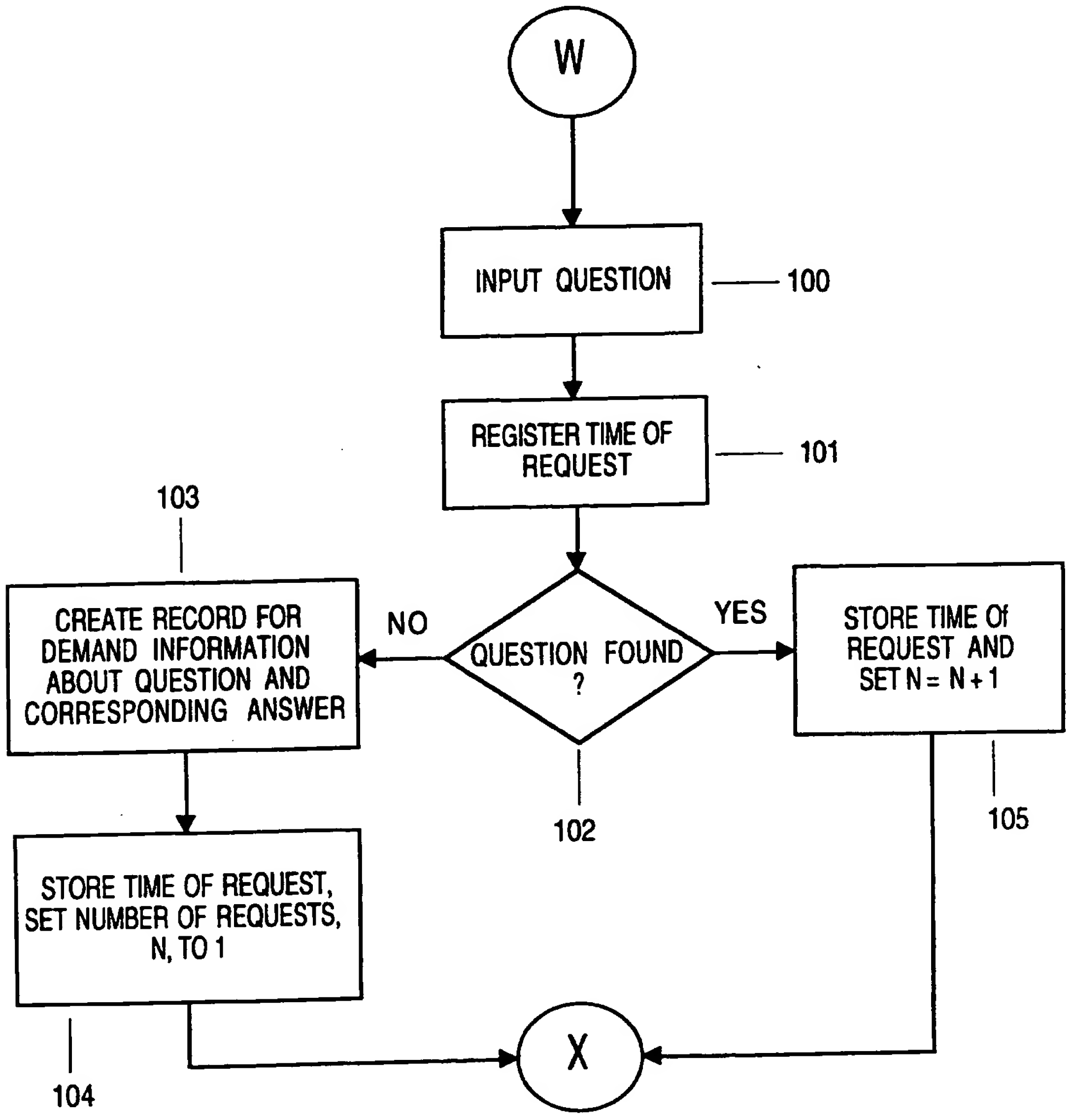
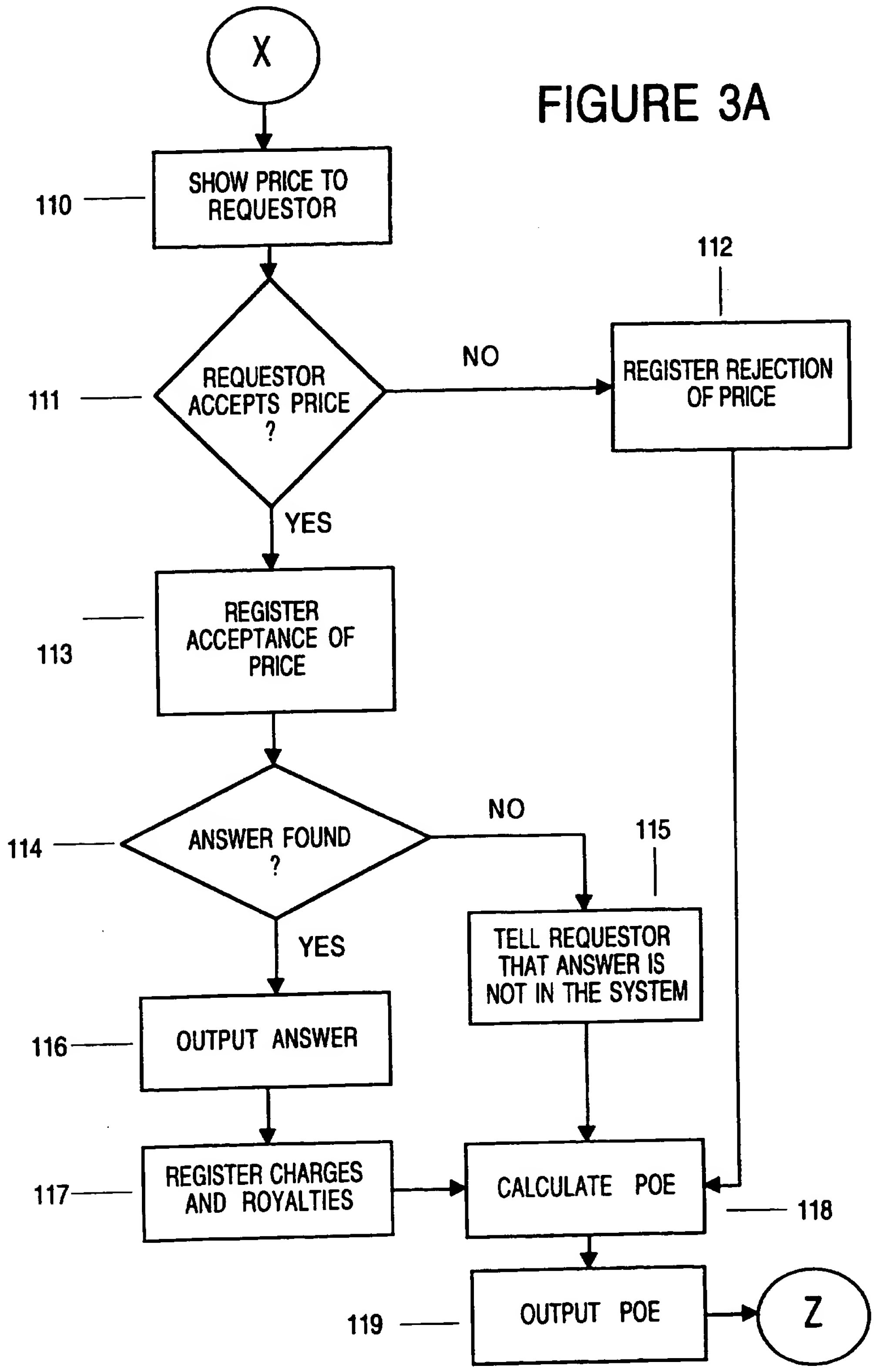

W
INPUT QUESTION
100
REGISTER TIME OF REQUEST
101
103
CREATE RECORD FOR DEMAND INFORMATION ABOUT QUESTION AND CORRESPONDING ANSWER
NO
QUESTION FOUND ?
YES
STORE TIME Of REQUEST AND SET N = N + 1
102
105
STORE TIME OF REQUEST, SET NUMBER OF REQUESTS, N, TO 1
104
X

FIGURE 3

FIGURE 3A


X
SHOW PRICE TO REQUESTOR
110
REQUESTOR ACCEPTS PRICE ?
111
NO
REGISTER REJECTION OF PRICE
112
YES
REGISTER ACCEPTANCE OF PRICE
113
ANSWER FOUND ?
114
NO
115
TELL REQUESTOR THAT ANSWER IS NOT IN THE SYSTEM
YES
OUTPUT ANSWER
116
REGISTER CHARGES AND ROYALTIES
117
CALCULATE POE
118
OUTPUT POE
119
Z

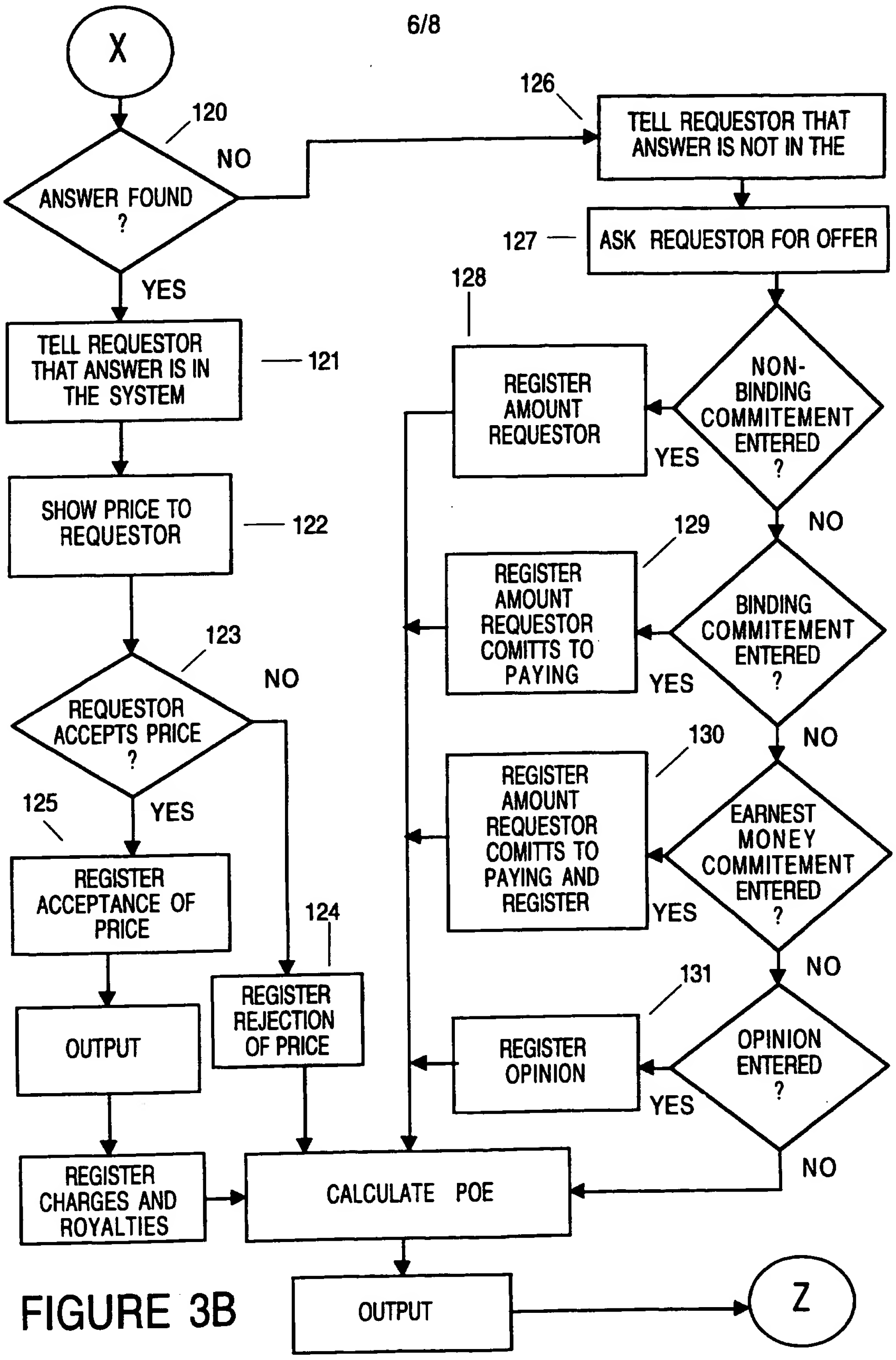

X
120
ANSWER FOUND ?
NO
YES
126
TELL REQUESTOR THAT ANSWER IS NOT IN THE
127
ASK REQUESTOR FOR OFFER
121
TELL REQUESTOR THAT ANSWER IS IN THE SYSTEM
122
SHOW PRICE TO REQUESTOR
123
REQUESTOR ACCEPTS PRICE ?
NO
YES
125
REGISTER ACCEPTANCE OF PRICE
124
REGISTER REJECTION OF PRICE
OUTPUT
REGISTER CHARGES AND ROYALTIES
128
REGISTER AMOUNT REQUESTOR
NON-BINDING COMMITEMENT ENTERED ?
YES
NO
129
REGISTER AMOUNT REQUESTOR COMITTS TO PAYING
BINDING COMMITEMENT ENTERED ?
YES
NO
130
REGISTER AMOUNT REQUESTOR COMITTS TO PAYING AND REGISTER
EARNEST MONEY COMMITEMENT ENTERED ?
YES
NO
131
REGISTER OPINION
OPINION ENTERED ?
YES
NO
CALCULATE POE
OUTPUT
Z

FIGURE 3B

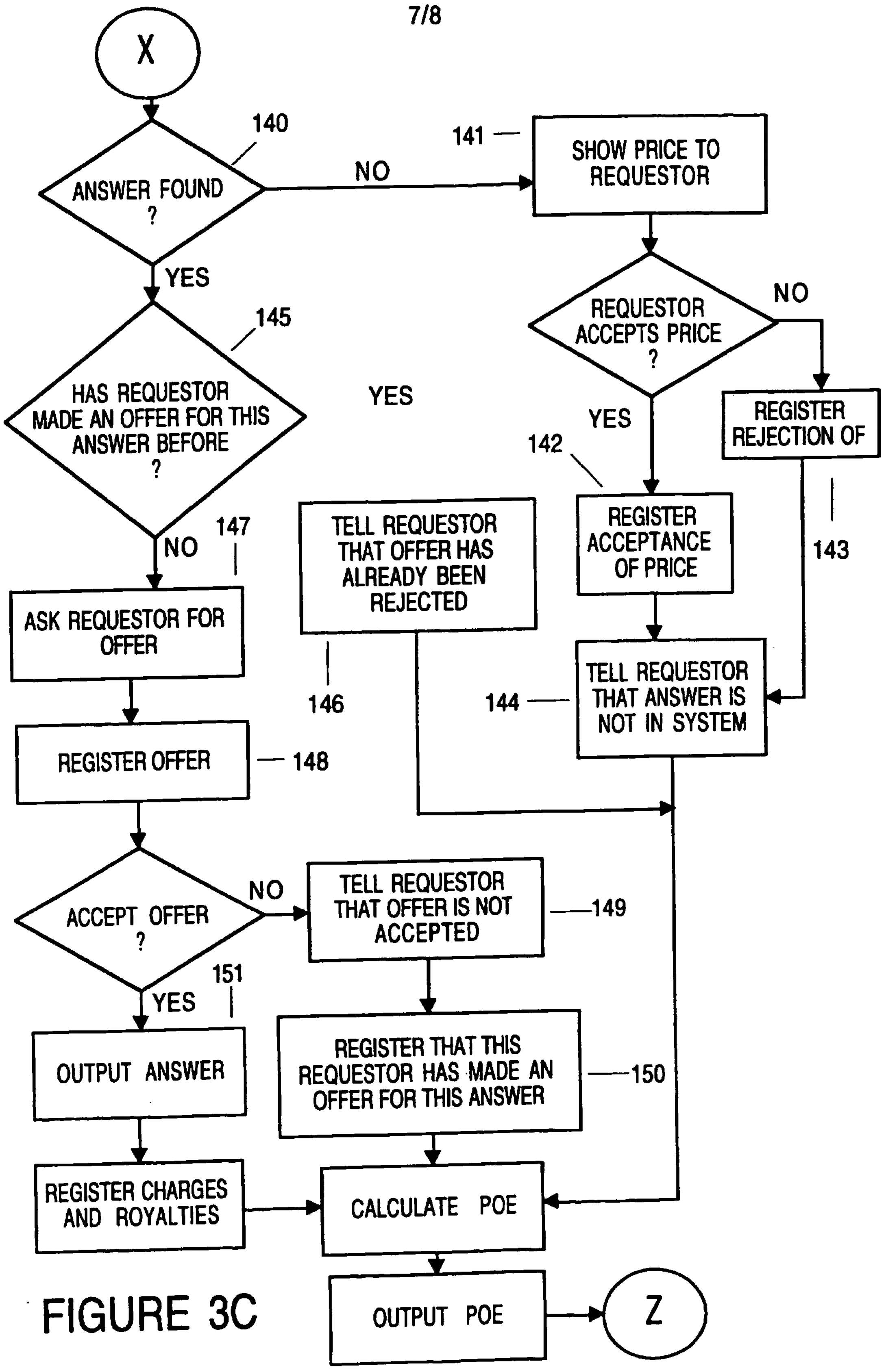

X
140
ANSWER FOUND ?
NO
141
SHOW PRICE TO REQUESTOR
YES
145
HAS REQUESTOR MADE AN OFFER FOR THIS ANSWER BEFORE ?
YES
REQUESTOR ACCEPTS PRICE ?
NO
YES
142
REGISTER REJECTION OF
REGISTER ACCEPTANCE OF PRICE
143
TELL REQUESTOR THAT OFFER HAS ALREADY BEEN REJECTED
147
NO
ASK REQUESTOR FOR OFFER
146
144
TELL REQUESTOR THAT ANSWER IS NOT IN SYSTEM
REGISTER OFFER
148
ACCEPT OFFER ?
NO
TELL REQUESTOR THAT OFFER IS NOT ACCEPTED
149
151
YES
OUTPUT ANSWER
REGISTER THAT THIS REQUESTOR HAS MADE AN OFFER FOR THIS ANSWER
150
REGISTER CHARGES AND ROYALTIES
CALCULATE POE
OUTPUT POE
Z

FIGURE 3C

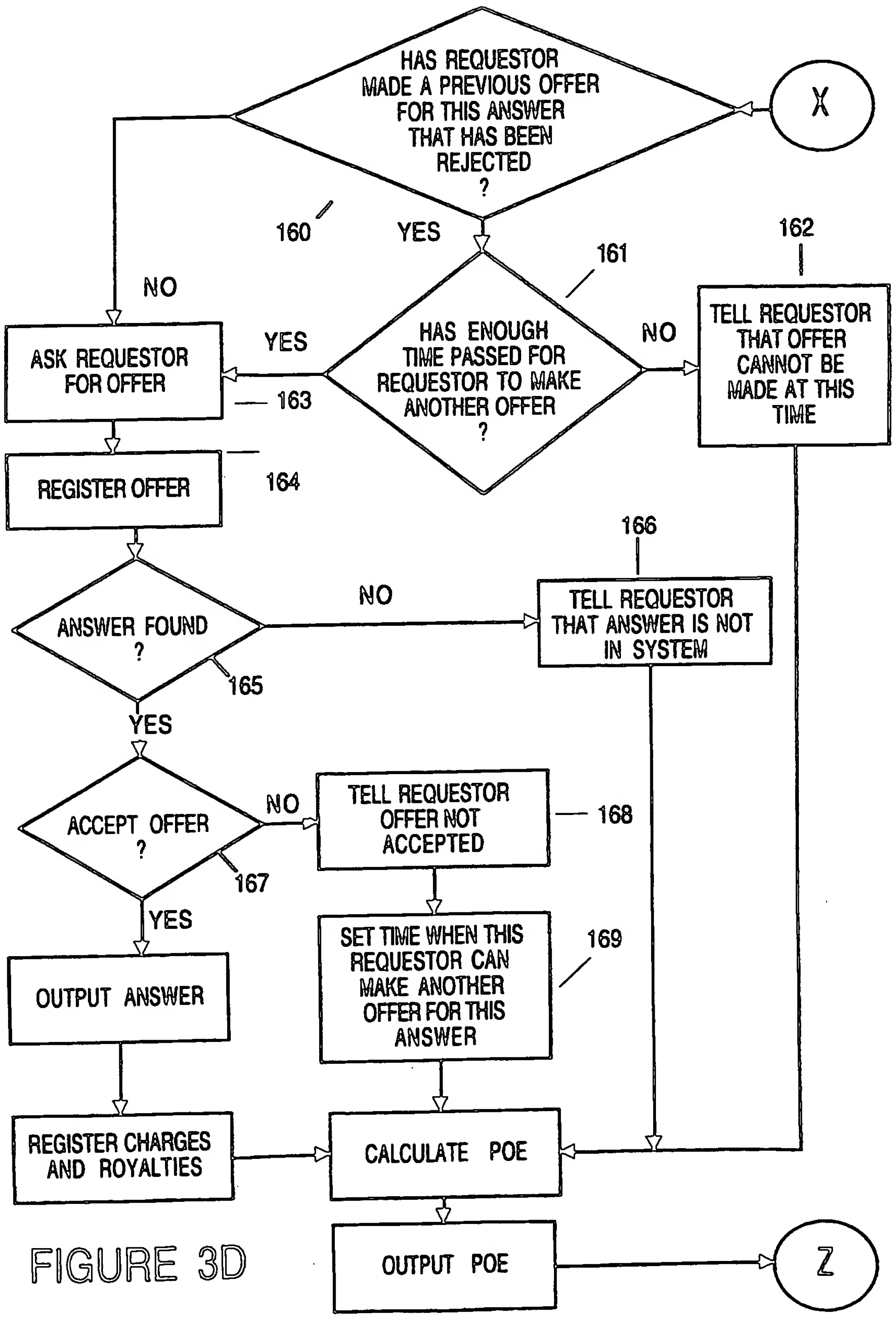

HAS REQUESTOR MADE A PREVIOUS OFFER FOR THIS ANSWER THAT HAS BEEN REJECTED ?
X
160
YES
NO
161
162
HAS ENOUGH TIME PASSED FOR REQUESTOR TO MAKE ANOTHER OFFER ?
YES
NO
TELL REQUESTOR THAT OFFER CANNOT BE MADE AT THIS TIME
ASK REQUESTOR FOR OFFER
163
REGISTER OFFER
164
166
ANSWER FOUND ?
NO
TELL REQUESTOR THAT ANSWER IS NOT IN SYSTEM
165
YES
ACCEPT OFFER ?
NO
TELL REQUESTOR OFFER NOT ACCEPTED
168
167
YES
SET TIME WHEN THIS REQUESTOR CAN MAKE ANOTHER OFFER FOR THIS ANSWER
169
OUTPUT ANSWER
REGISTER CHARGES AND ROYALTIES
CALCULATE POE
OUTPUT POE
Z

FIGURE 3D

INTERNATIONAL SEARCH REPORT

International application No.
PCT/US95/12630

A. CLASSIFICATION OF SUBJECT MATTER

IPC(6) :G06F 15/00; H04M 1/64
US CL :364/401
According to International Patent Classification (IPC) or to both national classification and IPC

B. FIELDS SEARCHED

Minimum documentation searched (classification system followed by classification symbols)

U.S. : 364/401, 402; 379/112, 114, 121, 126

Documentation searched other than minimum documentation to the extent that such documents are included in the fields searched

Electronic data base consulted during the international search (name of data base and, where practicable, search terms used)

Please See Extra Sheet.

C. DOCUMENTS CONSIDERED TO BE RELEVANT

Category°	Citation of document, with indication, where appropriate, of the relevant passages	Relevant to claim No.
A	US, A, 4,843,562 (KENYON ET AL) 27 June 1989, col. 1, lines 14-28 & col. 6, lines 59-65	1
A	US, A, 5,003,584 (BENYACAR ET AL.) 26 March 1991, abstract, figures 1 & 8-10	1
A	US, A, 5,101,352 (REMBERT) 31 March 1992, abstract, figure 4, col. 8, lines 39-45; andcol. 34, line 64 to col. 35, line 3	1
A,P	US, A, 5,418,713 (ALLEN) 23 May 1995, abstract, figures 3-7, col. 9, lines 41-47, col. 11, lines 21-28, and col. 14, lines 5-13,	1

[✓ X] Further documents are listed in the continuation of Box C. [ ] See patent family annex.

°	Special categories of cited documents:	"T"	later document published after the international filing date or priority date and not in conflict with the application but cited to understand the principle or theory underlying the invention
"A"	document defining the general state of the art which is not considered to be part of particular relevance	"X"	document of particular relevance; the claimed invention cannot be considered novel or cannot be considered to involve an inventive step when the document is taken alone
"E"	earlier document published on or after the international filing date		
"L"	document which may throw doubts on priority claim(s) or which is cited to establish the publication date of another citation or other special reason (as specified)	"Y"	document of particular relevance; the claimed invention cannot be considered to involve an inventive step when the document is combined with one or more other such documents, such combination being obvious to a person skilled in the art
"O"	document referring to an oral disclosure, use, exhibition or other means		
"P"	document published prior to the international filing date but later than the priority date claimed	"&"	document member of the same patent family

Date of the actual completion of the international search 05 JANUARY 1996	Date of mailing of the international search report 06 FEB 1996
Name and mailing address of the ISA/US Commissioner of Patents and Trademarks Box PCT Washington, D.C. 20231 Facsimile No. (703) 305-3230	Authorized officer ROBERT A. WEINHARDT [illegible] Telephone No. (703) 305-9780

INTERNATIONAL SEARCH REPORT

International application No.
PCT/US95/12630

C (Continuation). DOCUMENTS CONSIDERED TO BE RELEVANT

Category*	Citation of document, with indication, where appropriate, of the relevant passages	Relevant to claim No.
A,E	US, A, 5,465,213 (ROSS) 07 November 1995, abstract, figures 1-4; col. 1, line 55 to col. 3, line 40; col. 5, lines 36-52; and col. 14, lines 37-52	1

INTERNATIONAL SEARCH REPORT

International application No.
PCT/US95/12630

B. FIELDS SEARCHED
Electronic data bases consulted (Name of data base and where practicable terms used):

APS

search terms: determining or evaluating royalty/royalties, user bids, proposals, database, & computer